Automation of the Hazard and Operability Method Using Ontology-based Scenario Causation Models

Vom Fachbereich Maschinenbau und Verfahrenstechnik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte

Dissertation

von

Johannes-Imanuel Single

geb. in Villingen-Schwenningen

D 386

Tag der Disputation:	08.02.2022
Dekan:	Prof. Dr.-Ing. Tilmann Beck
Vorsitzender:	Prof. Dr.-Ing. Erik von Harbou
Berichterstatter:	Prof. Dr.-Ing. Jürgen Schmidt
	Prof. Dr.-Ing. Jens Denecke
	Prof. Dr.-Ing. Naim Bajcinca

Danksagung

Diese Dissertation ist das Ergebnis mehrjähriger Forschungsarbeit, die ich während meiner Tätigkeit als Doktorand am CSE Center of Safety Excellence (CSE Institut) in Kooperation mit der Technischen Universität Kaiserslautern durchgeführt habe. An dieser Stelle möchte ich allen ein großes Dankeschön aussprechen, die mich im Rahmen meiner Promotionsarbeit fachlich und persönlich unterstützt haben.

Allen voran möchte ich Prof. Dr.-Ing. Jürgen Schmidt für die Möglichkeit danken, meine Doktorarbeit am CSE durchzuführen und für die zahlreichen fachlichen Hilfestellungen und umfangreichen Korrekturen von Publikationen und der Dissertation. In gleichem Maße geht mein Dank an Prof. Dr.-Ing. Jens Denecke, welcher mich ans CSE holte und sich stets Zeit für Diskussionen, fachlichen Austausch und Korrektur von Publikationen und der Dissertation nahm. Weiterhin geht mein Dank an Prof. Dr.-Ing. Naim Bajcinca für die Übernahme des Korreferats und Prof. Dr.-Ing. Erik von Harbou für die Übernahme des Vorsitzes der Prüfungskommission.

Ein weiterer großer Dank gilt meinem langjährigen Arbeitskollegen Alexander Böhme, M.Sc. der jederzeit ein offenes Ohr für fachliche Diskussionen und Tipps und Tricks bezüglich Word bereithielt. Weiterer Dank gilt den von mir betreuten Bachelor- und Masteranden, die gerade in der Anfangsphase der Promotionsarbeit, einen wichtigen Beitrag geleistet haben. Zusätzlich möchte ich den (ehemaligen) Kollegen am CSE für die gemeinsame Zeit danken vor allen den Kollegen aus der Wangerooge Crew und allen weiteren exzellenten Kollegen.

Mein tiefster Dank gilt meiner Partnerin Julia Höhl, M.Sc., die mich durch alle Höhen und Tiefen der Promotionsarbeit begleitete und über Jahre hinweg unerschöpflich Verständnis für meine erhöhte Arbeitsbelastung und Geduld mit meiner Fokussierung auf das Thema aufbrachte.

09.02.2022
Johannes Single

Kurzzusammenfassung

Die HAZOP-Methode (Hazard and Operability) wird in der chemischen Prozessindustrie häufig zur Identifikation und Bewertung von Gefahren eingesetzt. Aufgrund ihrer menschenzentrierten Natur ist sie zeitaufwändig und die Ergebnisse hängen von der Zusammensetzung des Teams ab. Darüber hinaus beeinflussen die Faktoren Zeitdruck, Art der Durchführung, Erfahrung der Teilnehmer und Einbeziehung der Teilnehmer die Ergebnisse. Diese Forschungsarbeit zielt darauf ab, die HAZOP-Methode zu digitalisieren. Die Untersuchungen zeigen, dass sich wissensbasierte Systeme in Verbindung mit Ontologien zur Wissensrepräsentation eignen, um dieses Ziel zu erreichen. Zur Erfüllung dieser Aufgabe muss komplexes interdisziplinäres Wissen über Anlagen, Prozesse, Anlagenteile, Substanzen und den Standort dargestellt werden. Ein Ergebnis dieser Arbeit ist eine Taxonomie zur Klassifizierung von Anlagenteilen und eine entwickelte objektorientierte Equipmententitäten Bibliothek. Während der Entwicklung der Ontologie wurden typische HAZOP-Szenarien sowie deren Struktur, Komponenten und das zugrunde liegende Kausalmodell untersucht. Auf der Grundlage dieser Untersuchungen wurden semantische Beziehungen zwischen den Szenariokomponenten identifiziert. Im Rahmen einer automatischen Risikobewertung werden die Eintrittswahrscheinlichkeiten der Ursachen und das Schadensausmaß der Konsequenzen basierend auf einer Risikomatrix ermittelt, um Schutz- und Gegenmaßnahmen zuverlässig zu bestimmen. Zur Nutzbarmachung der Ontologie wird ein entwickelter Inferenzalgorithmus verwendet, welcher auf semantischen Reasonern und fallbasiertem Schließen basiert. Unter Berücksichtigung der Topologie wird die Fortpflanzung von Teilszenarien durch Anlagenteile berücksichtigt. Die Ergebnisse des entwickelten wissensbasierten Systems sind automatisch generierte HAZOP-Tabellen. Die Bewertung der erzielten Ergebnisse erfolgte anhand repräsentativer Fallstudien, in denen die Relevanz, Nachvollziehbarkeit und Vollständigkeit der automatisch identifizierten Szenarien betrachtet wird. Unter besonderer Berücksichtigung der kausalen Zusammenhänge der Szenarien, der Risikobewertung und der Schutzmaßnahmen ist die Qualität der automatisch generierten Ergebnisse mit konventionellen HAZOP-Tabellen vergleichbar. Die Untersuchungen zeigen, dass formale Ontologien geeignet sind, komplexes interdisziplinäres Wissen im Bereich der Prozess- und Anlagensicherheit darzustellen und um die HAZOP-Methode zu digitalisieren. Richtig eingesetzt, können wissensbasierte Systeme dazu beitragen, die Vorbereitungszeit und den repetitiven Charakter von HAZOP-Studien zu verringern und Ergebnisse zu standardisieren.

Abstract

The hazard and operability (HAZOP) method is widely used in chemical and process industries to identify and evaluate hazards. Due to its human-centered nature, it is time-consuming, and the results depend on the team composition. In addition, the factors time pressure, type of implementation, experience of the participants, and participant involvement affect the results. This research aims to digitize the HAZOP method. The investigation shows that knowledge-based systems with ontologies for knowledge representation are suitable to achieve the objective. Complex interdisciplinary knowledge regarding facility, process, substance, and site information must be represented to perform the task. A result of this work is a plant part taxonomy and a developed object-oriented equipment entity library. During ontology development, typical HAZOP scenarios, as well as their structure, components, and underlying causal model, were investigated. Based on these observations, semantic relationships between the scenario components were identified. The likelihood of causes and severity of consequences were determined as part of an automatic risk assessment using a risk matrix to determine safeguards reliably. An inference algorithm based on semantic reasoners and case-based reasoning was developed to exploit the ontology and evaluate the input data object containing the plant representation. With consideration given to topology, aspects like the propagation of sub-scenarios through plant parts were considered. The results of the developed knowledge-based system were automatically generated HAZOP worksheets. Evaluation of the achieved results was based on representative case studies in which the relevance, comprehensibility, and completeness of the automatically identified scenarios were considered. The achieved results were compared with conventionally prepared HAZOP tables for benchmark purposes. By paying particular attention to the causal relationships between scenario components, the risk assessment, and with consideration of safeguards, the quality of the automatically generated results was comparable to conventional HAZOP worksheets. This research shows that formal ontologies are suitable for representing complex interdisciplinary knowledge in the field of process and plant safety. The results contribute to the use of knowledge-based systems for digitizing the HAZOP method. When used correctly, knowledge-based systems can help decrease the preparation time and repetitious nature of HAZOP studies and standardize results.

List of contents

List of Tables ..... **XI**
List of Figures ..... **XII**
List of Symbols ..... **XIV**
List of Publications ..... **XV**
List of attended conferences ..... **XVI**
List of supervised student theses related to this dissertation ..... **XVII**
List of Abbreviations ..... **XVIII**
1. Introduction **1**
1.1 State of research 4
2. Aim, objectives, and overview **5**
3. Fundamentals **7**
3.1 Hazard identification methods 7
3.2 HAZOP methodology 8
3.2.1 HAZOP terminology 8
3.2.2 Procedure 10
3.2.3 Model concept of the HAZOP method 11
3.2.4 Concept map of working concepts 12
3.2.5 Team composition and competence 12
3.2.6 Risk assessment as part of the HAZOP study 13
3.2.7 Documentation of the study results 14
3.2.8 Other applications of the HAZOP method 15
3.3 Evaluation of the HAZOP method 15
3.4 Overview of graph theory 19
3.5 Data, information, and knowledge 19
3.6 From terms, concepts, and things to knowledge models 20
3.7 Knowledge representation formalisms 21
3.7.1 Advantages and disadvantages of knowledge representation formalism 22
3.8 Ontologies 24
3.8.1 Ontology languages 25
3.8.2 Web ontology language (OWL) 26
3.8.3 The logical foundation of selected ontology languages 28
3.8.4 Comparing ontology languages 28
3.9 Classification of reasoning techniques 31
3.9.1 Rule-based reasoning 32
3.9.2 Deductive classifiers 32

3.9.3 Case-based reasoning 33
3.9.4 Data-driven reasoning 33
3.9.5 Advantages and disadvantages of reasoning techniques 34
3.10 Summary of the fundamentals 35
4. Evaluation of the state of research 37
4.1 Approaches from the literature 37
4.1.1 HAZOPEX and HAZOPTool 37
4.1.2 Qualitative Hazard Identifier 37
4.1.3 ExpHAZOP+ 38
4.1.4 PHASuite 39
4.1.5 AutoHAZID 40
4.1.6 PetroHAZOP (HAZOPSuite) 41
4.1.7 LDGHAZOP 42
4.1.8 MFM HAZOP Assistant 43
4.1.9 D-higraph HAZOP assistant 44
4.1.10 Other Approaches 44
4.1.11 Ontologies in the field of process and plant safety 47
4.2 Summary of the evaluation of the state of research 48
4.3 Gaps in Research 49
5. Development of a procedure for computer-aided HAZOPs 51
5.1 Further requirements 51
5.2 Overview of the approach and principal considerations 52
5.2.1 Overview of the Ontology development process 53
5.2.2 Narrowing the knowledge domain 54
5.2.3 Investigation of causation models and core HAZOP concepts 55
5.2.4 Competency questions and domain-specific requirements 58
5.2.5 Naming convention 59
5.2.6 Modularization of the ontology 59
5.3 Conceptualization of the HazopCoreConcept 60
5.4 Substance entity 62
5.5 Equipment entity 65
5.5.1 Development steps in terms of abstraction and level of detail 66
5.5.2 The selected, overall level of detail and abstraction and structural plant items 67
5.5.3 Assembling equipment entities from structural plant items 70
5.5.4 The selected level of detail and abstraction – Functional plant items 71
5.5.5 Intended function 72
5.5.6 Mode of operation 74
5.5.7 Control instance 74

5.5.8 Boundary conditions 74
5.5.9 Ports 75
5.5.10 Site information 75
5.5.11 Assembling complete equipment entities 75
5.6 Equipment entity modeling 76
5.7 System modeling 82
5.8 Conceptualization of complementary semantic relationships 83
5.8.1 Embedded relationships 85
5.8.2 Semantic relationships of causes and underlying causes 86
5.8.3 Semantic relationships of effects and consequences 87
5.8.4 Semantic relationships of likelihood, severity, and risk 88
5.8.5 Semantic relationships of safeguards 89
5.9 Modeling of application-oriented ontology classes 91
5.9.1 Examples of formal class definitions 91
5.9.2 Event sequence within scenarios 93
5.9.3 Determination of the likelihood, severity, and risk 95
5.9.4 Programmatic creation of ontology classes and example class definitions 97
5.9.5 Importance of the boundary conditions and site information 98
5.10 Stages of development of ontology 101
5.10.1 Covering typical equipment types 102
5.11 Ontology learning 103
5.12 Equipment-based scenario identification 104
5.12.1 Inference strategy: equipment-based evaluation 104
5.12.2 Inference strategy: consequence sequence 106
5.13 Identification of subsequent deviations 107
5.13.1 Case-based reasoning for backward reasoning 109
5.13.2 The rationale for case-based reasoning 113
5.14 Propagation of hazards and operability issues 114
5.14.1 Inference strategy: process plant section topology 116
5.14.2 Algorithm for the identification of plant topologies 117
5.14.3 Propagation of deviations using propagation stacks 119
5.15 The architecture of the developed knowledge-based system 121
5.16 Procedure for the application of the proposed method 121
5.17 Summary of the developed knowledge-based system 123
6. Validation of the developed procedure 125
6.1 Case study I: Hexane storage tank 125
6.1.1 Model representation of the hexane tank 126
6.1.2 Automatically generated HAZOP results regarding the hexane tank 128

6.1.3 Reference HAZOP worksheet of the hexane tank ... 133
6.1.4 Differences and similarities in content and quality ... 133
6.2 Case study II: Water separator system ... 138
6.2.1 Model representation of the water separator system ... 138
6.2.2 Automatically generated HAZOP results regarding the olefin feed section ... 140
6.2.3 Propagation based analysis of the olefin feed section ... 143
6.2.4 Original HAZOP results of the olefin feed section ... 145
6.2.5 Differences in content and quality ... 145
6.3 Summary of the case studies ... 151
7. Discussion ... 153
7.1 Recap and implications of the literature review ... 153
7.2 Method-specific implications and significant findings ... 154
7.2.1 Decomposition of scenarios into their components ... 154
7.2.2 The selected level of abstraction of equipment and the environment ... 156
7.2.3 Ontology class and property definitions ... 157
7.2.4 Overlapping class definitions during risk assessment ... 160
7.2.5 Ontology evaluation using ontology metrics ... 161
7.2.6 Importance of topology-specific propagation and subsequent deviation ... 163
7.3 Relationships between findings and previous approaches ... 164
7.3.1 Safeguards and risk assessment ... 164
7.3.2 Propagation of events ... 165
7.3.3 Evaluation of the quality of automatically generated results ... 165
7.4 Socio-technical aspects and implications for the future ... 167
7.4.1 Trust in automation ... 167
7.4.2 Situational awareness and loss of skill ... 168
7.4.3 Implementing a knowledge-based system into the HAZOP process ... 169
7.4.4 Implications for HAZOP studies in the future ... 170
7.5 Limitations of the developed methodology ... 170
8. Conclusion ... 173
Literature ... 175
Appendix ... 189

List of Tables

Table 1: Typical guide-word and parameter combinations (cf. Crawley and Tyler, 2015b) 11
Table 2: Industry typical HAZOP worksheet 15
Table 3: Advantages and disadvantages of HAZOP studies I 18
Table 4: Advantages and disadvantages of knowledge representation formalisms 23
Table 5: Relationship between OWL constructors, DL and Manchester OWL syntax 28
Table 6: Comparison of the RDFS, OWL 1, and OWL 2 (cf. Munir and Sheraz Anjum, 2018) 31
Table 7: Advantages and disadvantages of different reasoning techniques 35
Table 8: Summary of the literature review 50
Table 9: Example distinctions between effects and consequences 57
Table 10: Example distinctions between underlying causes and causes 57
Table 11: Standards and approaches addressing the classification of plants 65
Table 12: Assembling equipment entities from structural plant items 71
Table 13: Site information 75
Table 14: Example relationships to other concepts and rationale for identifying the likelihood of causes 96
Table 15: Example relationships to other concepts for identifying the severity of consequences 96
Table 16: Jaro-Winkler distance of different case attributes 110
Table 17: Example cases for identifying subsequent deviations 112
Table 18: Excerpt from the Safety Data Sheet: Hexane 126
Table 19: Function description matrix of the hexane tank 128
Table 20: Automatically generated extended HAZOP worksheets of the hexane tank I 130
Table 21: Automatically generated extended HAZOP worksheets of the hexane tank II 132
Table 22: Reference HAZOP worksheet of the inflow line based on (CCPS, 2001) 134
Table 23: Reference HAZOP worksheet of the hexane tank based on (CCPS, 2001) 135
Table 24: Comparison of the identified safeguards in case study I 137
Table 25: Function description matrix of the olefin feed section 138
Table 26: Excerpt from the Safety Data Sheet: Hexene 140
Table 27: Automatically generated extended HAZOP worksheets of the olefin feed section I 142
Table 28: Automatically generated extended HAZOP worksheets of the olefin feed section II 144
Table 29: Reference HAZOP worksheet of the olefin feed section I (cf. Parmar and Lees, 1987) 148
Table 30: Reference HAZOP worksheet of the olefin feed section II (cf. Parmar and Lees, 1987) 149
Table 31: Comparison of the identified safeguards 150
Table 32: Comparison of ontology metrics 163
Table 33: Comparison of the form of presentation of different approaches 167
Table 34: Excerpt of HAZOP worksheets from the ExpHAZOP$^+$ approach 167

List of Figures

Figure 1: Transition from a classic to digital HAZOP process 2
Figure 2: Most frequently used hazard identification methods (cf. Mock and van Mahnen 1998) 8
Figure 3: Schematic representation of the HAZOP process 10
Figure 4: Different model conceptions to describe incidents/accidents 12
Figure 5: Concept map of working concepts 13
Figure 6: Risk matrix based on Schwarz et al. (2019) 13
Figure 7: Causality model behind the HAZOP model 16
Figure 8: Comparison of industrial practice and intermediate event idea 17
Figure 9: Exemplary simple graph 19
Figure 10: Relationship between data, information and knowledge 20
Figure 11: The semiotic triangle applied to a fire 21
Figure 12: Overview of knowledge representation formalisms (cf. Hoekstra, 2009) 22
Figure 13: Example of a triple 26
Figure 14: Classification of relevant ontology languages 26
Figure 15: Essential elements of the OWL ontology language 27
Figure 16: Disjointness between classes 29
Figure 17: Usage of ontology languages 2006/2007 (cf. Cardoso, 2007) 30
Figure 18: Number of hits on Google Scholar (query on 28.11.2020) 31
Figure 19: Overview of reasoning techniques 32
Figure 20: Subsumption explained 33
Figure 21: Schematic two-layered reasoning engine (cf. Single et al., 2019b) 40
Figure 22: LDG model of three guidewords and three variables (cf. Single et al., 2019b) 43
Figure 23: Exemplary MFM model (cf. Single et al., 2019b) 43
Figure 24: Exemplary D-higraph system (cf. Single et al., 2019b) 45
Figure 25: Overview of the method developed for the automation of HAZOPs 52
Figure 26: Related knowledge fragments within HAZOP worksheets 55
Figure 27: Causal sequence of events of scenarios 56
Figure 28: Enhanced concept map of working concepts 58
Figure 29: Modularization of the ontology 60
Figure 30: Relationships for the generation of deviations 61
Figure 31: Concept map of the HazopCoreConcept 62
Figure 32: a) Overlapping of class definitions without considering the substance task b) Unambiguous class definitions under consideration of substance tasks 63
Figure 33: Illustration of the classes and relations of the substance concept. 64
Figure 34: Classification of terms using a P&ID section 66
Figure 35: Assembling equipment entities based on subsystems 67
Figure 36: Classification of physical assets in different standards and approaches 68
Figure 37: Development stages of the equipment entity representation 69
Figure 38: Selected hierarchical structure for representing plants 69
Figure 39: Taxonomy of the relevant structural plant items 71
Figure 40: Taxonomy (class hierarchy) of the relevant functional plant items 73
Figure 41: Illustration of the classes and relations of the equipment entity concept. 77
Figure 42: Assembling the equipment entity: cooled storage tank 77
Figure 43: Assembling the equipment entity: external floating roof tank 78
Figure 44: Assembling the equipment entity: wet scrubber 79
Figure 45: Assembling the equipment entity: shell and tube heat exchanger 79
Figure 46: Assembling the equipment entity: air-cooled condenser 80

Figure 47: Assembling the equipment entity: centrifugal pump 81
Figure 48: Assembling the equipment entity: control valve 81
Figure 49: Representation of an example plant section 84
Figure 50: Overall ontological structure 85
Figure 51: Explicit and embedded relationships 85
Figure 52: Concept map of cause-related semantic relationships 87
Figure 53: Concept map of effect and consequence-related semantic relationships 88
Figure 54: Concept map of likelihood, severity, and risk-related semantic relationships 89
Figure 55: Concept map of safeguard-related semantic relationships 90
Figure 56: Partial cause-effect chain concerning a centrifugal pump 94
Figure 57: Partial cause-effect chain concerning a storage tank 95
Figure 58: Description of the components of the formal class definitions of the ontology 97
Figure 59: Stages in the ontology development process 102
Figure 60: Extent of the modeled scenario components 103
Figure 61: Ontological model described in Single et al. (2020b) 104
Figure 62: Inference algorithm: equipment-based mode 106
Figure 63: Sub function: consequence cascade and safeguards 107
Figure 64: References between scenarios within a typical HAZOP worksheet 108
Figure 65: Forward and backward reasoning 108
Figure 66: Partial cause-effect chain concerning a runaway reaction 108
Figure 67: Influence of attribute weighting on the threshold 112
Figure 68: Example of an inverse property including queries and query answers 113
Figure 69: Overview propagation and equipment-based analysis 115
Figure 70: Basic plant section topologies 116
Figure 71: Signed directed graph representation of an excerpt of the CSE High-Pressure-Loop highlighted basic plant section topologies 117
Figure 72: Inflow/outflow ratios of different topologies 118
Figure 73: Flow chart of the plant section identification algorithm 118
Figure 74: P&ID of the plant section 119
Figure 75: Graph representation of the original plant section and decomposed plant section 120
Figure 76: Example propagation of a partial scenario 120
Figure 77: Example propagation paths within the settling tank system 121
Figure 78: Architecture of the knowledge-based system and interaction with humans 122
Figure 79: P&ID of the Hexane tank 125
Figure 80: Object diagram of the hexane storage tank model 127
Figure 81: SDG representation of the tank truck, pump and hexane tank 128
Figure 82: Feed section of the olefin dimerization plant (cf. Parmar and Lees, 1987) 138
Figure 83: SDG representation of the feed section of olefin dimerization plant 139
Figure 84: SDG representation and decomposition of the feed section 145
Figure 85: Components of four different scenarios involving a storage tank 155
Figure 86: Example of the property axiom: AsymmetricProperty 159

List of Symbols

A, B, C, D, E	Classes (or concepts) in the context of formal logics
a_{Ai}	Case attribute i of case A
a_{Bi}	Case attribute i of case B
AP	Average population of an ontology in the context of ontology metrics
C_{tot}	Total number of classes of an ontology in the context of ontology metrics
CR	Average population of an ontology in the context of ontology metrics
e_{ij}	An edge between node i and j
$E(G)$	Edge set of graph G
G	Graph in a graph theoretical context
H	Inheritance relationships in the context of ontology metrics
I	Input function
I	Number of instances (individuals) in an ontology
IR	Inheritance richness in the context of ontology metrics
K	Datatype in the context of formal logics
O	Output function
$N(G)$	Node set of graph G
P	Set of places
P	Non-inheritance relationships in the context of ontology metrics
r	Object property (or role) in the context of formal logics
RR	Relationship richness in the context of ontology metrics
s	Local similarity measure between case attributes
$S_T(A, B)$	Total similarity measure between cases A and B
T	Set of transitions
v	Data property (or role) in the context of formal logics
w_i	Weight of case attribute i
$\equiv$	Equivalence relation
$\sqcap$	Intersection of a number of sets (set theory)
$\sqcup$	Union of a number of sets (set theory)
$\exists$	Existential restriction in the context of formal logics

List of Publications

This dissertation is based on the following publications.

I. J. I. Single, J. Schmidt, J. Denecke, State of research on the automation of HAZOP studies, Journal of Loss Prevention in the Process Industries, Volume 62, 103952, 2019. https://doi.org/10.1016/j.jlp.2019.103952

II. J. I. Single, J. Schmidt, J. Denecke, Computer-aided HAZOP studies: knowledge representation and algorithmic hazard identification, WIT Transactions on the Built Environment, Volume 189, pp. 55-66, 2019. https://doi.org/10.2495/SAFE190061

III. J. I. Single, J. Schmidt, J. Denecke, Knowledge acquisition from chemical accident databases using an ontology-based method and natural language processing, Safety Science, Volume 129, 104747, 2020. https://doi.org/10.1016/j.ssci.2020.104747

IV. J. I. Single, J. Schmidt, J. Denecke, Ontology-Based Support for Hazard and Operability Studies, International Journal of Safety and Security Engineering, Volume 10(3), pp. 311-319, 2020. https://doi.org/10.18280/ijsse.100302

V. J. I. Single, J. Schmidt, J. Denecke, Computer-Aided Hazop: Ontologies and Ai for Hazard Identification and Propagation, Proceedings of the 30th European Symposium on Computer Aided Process Engineering (ESCAPE30), S. Pierucci, F. Manenti, G. Bozzano (Eds.), May 24-27, 2020. https://dx.doi.org/10.1016/B978-0-12-823377-1.50298-6

VI. J. I. Single, J. Schmidt, J. Denecke, Ontology-based computer aid for the automation of HAZOP studies, Journal of Loss Prevention in the Process Industries, Volume 68, 104321, 2020. https://doi.org/10.1016/j.jlp.2020.104321

List of attended conferences

I. Single et al., Expert System for Risk Analysis of Technical Systems, Tagungsband Collaborative European Research Conference (CERC), Karlsruhe, 22. September 2017

II. Single et al., Risikoabschätzung durch Expertensysteme – brauchen wir den Experten noch?, 56. Tutzing Symposium, Tutzing, September 25-27, 2017

III. Single et al., ExpertRisk - Expertensysteme für Risikoanalysen technischer Anlagen, CSE-Sicherheitstage, Wangerooge, April 20-22, 2018

IV. Single et al., ExpertRisk - Computer-aided Hazard Identification for Process Plants, 3rd Joint Meeting of the US and European DIERS User Group, Ettlingen, May 22-24, 2019

V. Single et al., Computer-aided HAZOP studies - knowledge representation and algorithmic hazard identification, SAFE19 conference, Ancona, Italy, September 23-25, 2019

VI. Single et al., Computer-aided Hazop: Ontologies and Ai for Hazard Identification and Propagation, European Symposium of Computer-Aided Process Engineering, ESCAPE 30 VIRTUAL SYMPOSIUM, August 31- September 2, 2020

VII. Single et al., On the usage of ontologies for the automation of HAZOP studies, 2020 MARY KAY O'CONNOR Virtual Process Safety Symposium, Oktober 20-21, 2020

List of supervised student theses related to this dissertation

I. G. Römling, Entwicklung eines Expertensystems zur Risikoanalyse von chemischen und petrochemischen Anlagen, Master's thesis, KIT – Karlsruhe Institute of Technology, 2017.

II. O. Schmid, Entwicklung einer Methodik zur Durchführung von Risikoanalysen am Beispiel einer Ammoniakkälteanlage, Bachelor's thesis, University of Applied Sciences Karlsruhe, 2017.

III. M. Zimmer, Entwicklung eines Expertensystems zur Durchführung von Risikoanalysen am Beispiel einer Ammoniakkälteanlage, Master's thesis, KIT – Karlsruhe Institute of Technology, 2018.

List of Abbreviations

AHA	Automatic hazard analyzer
AI	Artificial Intelligence
AP	Average population (ontology metric)
API	Application Programming Interface
ASME	American Society of Mechanical Engineers
ATEX	Atmosphères Explosibles
BEL	Bayesian Ontology Reasoner
BimSchG	Bundes-Immissionsschutzgesetz (Federal Immission Control Act)
BLEVE	Boiling Liquid Expanding Vapor Explosion
BLHAZID	Blended Hazard Identification
BPCS PL	Basic Process Control System Protection Layer
CAD	Computer Aided Design
CAHO	Computer-aided HAZOP ontology
CAO	Chemical accident ontology
CAPE	Computer Aided Process Engineering
CAEX	Computer Aided Engineering Exchange
CAS	Chemical Abstracts Service
CBR	Case-Based Reasoning
CCPS	Center for Chemical Process Safety
CCF	Common Cause Failure
CQ	Competency question
CSE	Center of Safety Excellence
CL	Common Logic
CLIPS	C Language Integrated Production System
CR	Class richness
DAG	Directed Acyclic Graph
DAML+OIL	DARPA Agent Markup Language; Ontology Inference Layer
DIN	Deutsches Institut für Normung e. V. (German Institute for Standardization)
DL	Description Logics
ETA	Event Tree Analysis
FMEA	Failure Mode and Effect Analysis
FOL	First-Order Logic
FTA	Fault tree Analysis
GHS	Globally Harmonised System
HACCP	Hazard Analysis and Critical Control Points
HAZAN	Hazard analysis
HAZID	Hazard identification
HAZOP	Hazard and operability studies
HTML	Hypertext Markup Language
ICI	Imperial Chemical Industries
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission

IPL	Independent Protection Layer (onion skin model)
IR	Inheritance richness (ontology metric)
ISO	International Organization for Standardization
KB	Knowledge Base
KBS	Knowledge-Based System
KEE	Knowledge Engineering Environment
KIF	Knowledge Interchange Format
KL-ONE	Knowledge Representation System
KPI	Key Performance Indicator
KM	Knowledge Model
KR	Knowledge Representation
KROSA	Knowledge Reuse Oriented Safety Analysis
LDG	Layered Digraph Model
LOPA	Layer of Protection Analysis
LOPC	Loss of Primary Containment
LP	Logic Programming
MFM	Multilevel Flow Modeling
ML	Machine Learning
MOC	Management of Change
MSDS	Material Safety Data Sheet
NLP	Natural Language Processing
NPSH	Net Positive Suction Head
OOP	Object-Oriented Programming
OSHA	Occupational Safety and Health Administration
OWL	Web Ontology Language
P&ID	Piping and Instrumentation Diagram
PED	Pressure Equipment Directive
PFD	Process Flow Diagram
PFFM	Process Flow Failure Modes
PHA	Process Hazard Analysis
PLC	Programmable Logic Controller
PLM	Product Cycle Management
PTC	Performance Test Codes
QDE	Qualitative Differential Equations
QHI	Qualitative Hazard Identifier
QPC	Qualitative Physics Compiler
QSIM	Qualitative Simulation
RA	Risk Analysis
RBR	Rule-Based Reasoning
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RR	Relationship richness (ontology metric)
RuleML	Rule Markup Language
SADT	Structured Analysis and Design Technique

SDG	Signed Directed Graph
SDS	Safety Data Sheet
SHO	Schematic HAZOP ontology
SIL	Safety Integrity Level
SOM	Scenario Object Model
SQL	Structured Query Language
SRS	Safety Requirements Specification
STPA	Systems Theoretic Process Analysis
SVM	Support Vector Machine
SWRL	Semantic Web Rule Language
TRGS	Technische Regeln für Gefahrstoffe
TRL	Technology Readiness Level
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
ZHA	Zurich Hazard Analysis

1. Introduction

Within the chemical process industry, the safe operation of process plants is essential to protect human health and the environment. According to the Federal Immission Control Act (Bundes-Immisionsschutzgesetz, BImSchG), installations shall be established and operated in a way that harms to the environment or other hazards, considerable disadvantages, and considerable nuisance to the general public and neighborhood are avoided. To achieve this goal, engineers conduct safety assessments. Safety in this context means that hazardous events are prevented or damage is limited. A hazard can be described as "a physical or chemical condition that has the potential for causing harm to people, property, or the environment" (cf. CCPS, 2008b). Leveson further defined a hazard as "a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to a loss" (cf. Leveson and Thomas, 2018, pp. 17). Accordingly, a hazard can be understood as a threat that emanates from dangerous substances and physical conditions and represents a system state or set of conditions; however, hazards cannot be entirely eliminated but reduced to an acceptable minimum.

Process hazard analysis (PHA) methods are applied to identify and assess potential hazardous events within processes and process plants systematically. Legal requirements enforced by authorities must be considered during PHA alongside regulatory guidelines from companies. In accordance with the Seveso-III directive, the plant operator must take all necessary steps to prevent major accidents and mitigate their consequences. These steps include providing authorities with a safety report on major accident scenarios, risk analysis, prevention and intervention measures, and management systems. The directive comes into effect if hazardous substances are processed or produced in the plant in sufficient quantities.

In addition to maintaining plant safety, production availability can also be increased by identifying weak spots and defining preventive measures. Various methods are available for this purpose. One of the most used and recognized PHA methods is the hazard and operability (HAZOP) study. According to IEC 61882, HAZOP is a systematic technique for identifying risks and operability problems associated with the operation and maintenance of systems. In German-speaking areas, the HAZOP procedure is known as the PAAG procedure (**P**rognose, **A**uffinden der Ursachen, **A**bschätzen der Auswirkungen, **G**egenmaßnahmen; engl. prognosis, finding the causes, estimating the effects, countermeasures). A HAZOP study aims to identify and assess potential hazardous scenarios, equipment failures, operability problems, or quality issues that may arise from plant operation, transportation, or processes. It also includes a risk assessment, the definition of safeguards, documentation of results, and guidelines for further actions. HAZOP studies are conducted multiple times during the plant life cycle, e.g., during conceptual design, construction, operation phases, as well as during commissioning or in the context of process and plant modifications. It is a human-centered, guided brainstorming technique that is usually conducted by a team of interdisciplinary experts, including process, control, and safety engineers or other specialists. The HAZOP study leader is responsible for moderating and documenting the study results.

Recently, government regulators have been increasing their demands for public safety and environmental protection. These demands include safety assessments, which can take days to months to complete depending on the plant. These assessments are also impacted by the practitioner's personal experience, moderation, communication, discussion culture, and training. Furthermore, the study participants' abilities can be limited by the repetitive tasks, large amounts of data, and stressful conditions often associated with HAZOP studies. These challenges are further compounded by the current COVID-19 pandemic, which has led to HAZOP studies being held partly via web conferencing. This setup interrupts communication, as HAZOP participants would normally meet in person to share their experiences and may reduce their abilities to focus due to distractions associated with remote communication. In addition, not all participants may have all the information because they are not in the same room.

For years, efforts have been made to accelerate, streamline, and standardize the HAZOP process using computer systems. As such, the automation of HAZOP is frequently mentioned, and the literature refers to computer-aided HAZOP. These descriptions imply a computer system that automatically performs parts of the HAZOP study, such as hazard identification. Such computer systems could reduce human efforts and staff costs while compensating for HAZOP practitioners' limitations. For instance, the time expense could be reduced directly in the pre-work phase. Simultaneously, methodological disadvantages could be compensated, including the single root cause assumption or the disregard of failures not associated with guidewords. Furthermore, computer systems could contribute to the standardization and reproducibility of HAZOP results through the systematic identification and processing of scenarios. Generally, the results of the studies depend heavily on the professional experience and knowledge of the study participants.

Conversely, the results generated by computer systems could serve as a basis for discussion or even go beyond the team's experience. They could also assist engineers in training or non-experts participating in HAZOP studies while serving as a guideline. In addition, findings from previous HAZOP studies could be transferred into current studies. It is also worth noting that routine jobs could be automated, for example, by analyzing standard plant equipment such as storage tanks, pump systems, or utilities such as refrigeration systems. Taking over routine activities allows study participants to focus on more critical or non-routine plant sections. Additionally, reduced alertness among participants can be mitigated by reducing the documentation workload and time required.

Automation poses a challenge in transferring and digitizing the human-centered HAZOP process (cf. Figure 1). The primary source of information of HAZOP practitioners is piping and instrumentation diagrams (P&IDs), process flow diagrams (PFDs), and process descriptions. In contrast, a computer system requires a representation of this information as an input. In addition, the practitioner uses personal knowledge, training, professional experience, and company or authority guidelines, while a computer system requires a formal representation of this expert knowledge. Moreover, the practitioner concludes potential hazardous scenarios, operability issues, risks, and appropriate safeguards based on available information and personal knowledge. Within a computer system, conclusions are drawn, and process-safety information is evaluated using an inference algorithm to identify and assess scenarios and propose appropriate safeguards.

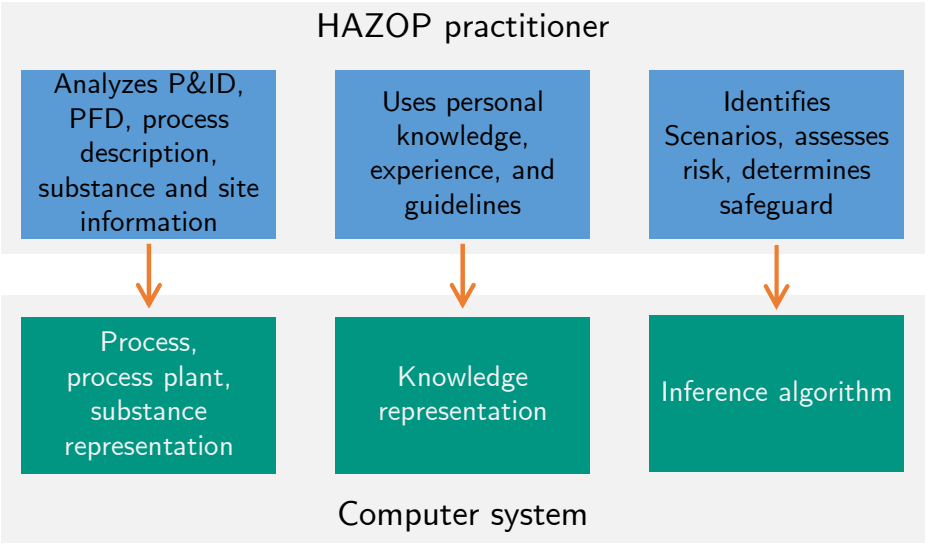


Figure 1: Transition from a classic to digital HAZOP process

HAZOP-relevant knowledge is present in explicit forms within HAZOP worksheets and implicit in the understanding and experiences of practitioners. The knowledge required to conduct HAZOP studies is interdisciplinary and covers process engineering, process control, plant engineering, and safety engineering, and various aspects of the different disciplines must be considered. Besides the interconnectedness between these disciplines, there is diversity among plant types and processes within the chemical process industry. This diversity results in a high degree of complexity of the domain to be represented that can only be reduced to a limited extent to avoid drawing false conclusions.

Human experts can cope with uncertain, imprecise, incomplete, or inconsistent information, while computational intelligence relies on deductive logic and is limited when information is incomplete (cf. Kern-Isberner et al., 2017). The challenge of developing such computer systems is evident when modeling the HAZOP process using an engineering approach and describing it using computer science methods. The main difficulty is that, due to the high degree of freedom, the required granularity and knowledge representation structure are unknown a priori. The causal models underlying the scenarios play a crucial role in automatically identifying scenarios. Lehto and Salvendy (1991) described accident causation models as the theoretical framework for guiding retrospective accident analysis and prospective hazard identification. According to Khan et al. (2015), accident models characterize the relationship between causes and consequences.

Typically, scenarios are identified based on nodes representing a plant section with a specified intended function in conventional HAZOP studies. Analogous to the HAZOP practitioner, who examines plant sections (nodes) based on P&IDs, PFDs, and a process description, the plant section to be examined must be represented and processed within a computer system. The difficulty here is finding an appropriate level of detail and abstraction since too coarse of a representation leads to a lack of information. At the same time, a too granular representation significantly increases the modeling effort without providing any significant added value. Identifying an appropriate level of detail is challenging due to the diversity and complexity of process plants within the chemical industry. Accordingly, Taylor (2017) noted that system modeling can become more costly than a human-led HAZOP study.

The dependencies between scenarios, equipment, processes, substances, intended function, mode of operation, site, and environment information are vast. The recognition and adaptation of scenarios and the consideration of these aspects are challenging and crucial for answering whether computer systems can achieve qualitatively comparable results compared to HAZOP practitioners. In addition to hazard identification, a risk assessment is typically conducted to focus on severe scenarios. Afterward, safeguards are specified to reduce the risk of a scenario to an acceptable level. Hence the extent and required availability of safeguards directly depend on the risk category of the scenario. For a computer system to automatically identify safeguards, the risk assessment must also be conducted automatically.

1.1 State of research

Automated HAZOP has been a research topic in academia for around 30 years. *Automation* can be applied in four areas: (1) information acquisition, (2) information analysis, (3) decision and action selection, and (4) action implementation (cf. Parasuraman et al., 2000). In the past, expert systems were often used to achieve this goal. Feigenbaum (1982), a pioneer in the field of expert systems, defined an expert system as "an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution." Styczynski et al. (2017) defined an expert system as a knowledge-based system with inference or problem-solving capabilities used in a particular domain. In ISO 1087, a field of specialized knowledge is referred to as a domain. An overview of existing approaches and prototypes for computer-aided HAZOP studies was provided by (Single *et al.*, 2019a). The most important approaches are summarized below.

One of the first approaches aimed to create fault trees automatically (cf. Taylor, 1982). Galluzzo et al. (1999) used logic trees to model causes and consequences. Early systems were mainly built on rules, such as an expert system for malfunction detection by Shum et al. (1988). Suokas et al. (1990) proposed the rule-based prototype HAZOPEX to identify potentially hazardous events. Other works, including HAZOPTOOL, were built on rule-based reasoning systems and various knowledge bases (cf. Heino et al., 1995). Multiple prototypes were built using rule networks for cause and consequence reasoning, such as TOPHAZOP, OptHAZOP, EXPERTOP, and ExpHAZOP$^+$ (cf. Chae and Yoon, 1994; Khan and Abbasi, 1997a; Khan and Abbasi, 1997b; Khan and Abbasi, 2000; Rahman *et al.*, 2009). Graf and Schmidt-Traub (2000) developed a model-based approach for hazard identification based on qualitative plant models formulated as statecharts. Catino and Ungar (1995) proposed a prototype called QHI, which was based on qualitative equations describing plant behavior and intended for automated hazard identification. Marx (2003) aimed to integrate rule-based hazard identification into CAD software. Park (2017) developed a method for hazard identification based on extracted P&ID information called DA-HAZID. (Schmidberger *et al.*, 2009) proposed the usage of CAEX data for knowledge-based support of HAZOP studies.

Suh, Lee and Yoon (1997) and Kang et al. (1999) introduced AHA, a multi-model approach using an expert system shell and a rule base for hazard identification. Many approaches used directed graphs combined with rule-based reasoning, such as HAZEXPERT, HAZOPExpert, AutoHAZID (cf. Göring and Schecker, 1993; Vaidhyanathan *et al.*, 1996; McCoy, Wakeman, Larkin, Jefferson, *et al.*, 1999; Wang *et al.*, 2009). Furthermore, Petri nets have been proposed for batch processes, including Batch HAZOPExpert by Srinivasan and Venkatasubramanian (1998) and PHASuite by Zhao *et al.* (2005a). Boonthum et al. (2014) combined structural models with directed graphs for hazard identification. Other researchers build their prototypes on more complex graph-based methods, such as LDGHAZOP, Functional HAZOP Assistant, and D-higraphs HAZOP Assistant (cf. Cui *et al.*, 2010; Rossing et al., 2009; Rossing et al., 2010; Rodríguez and De la Mata, 2012). Some of the more recent approaches used paradigms from artificial intelligence, such as case-based reasoning (cf. Zhao et al., 2009; Daramola et al., 2011). Few researchers considered safeguards at all (cf. Kang et al., 1999; Zhao et al., 2005a; Zhao et al., 2009; Daramola et al., 2013). Even when safeguards were considered, they could not be reliably identified (cf. McCoy et al., 2000). In the research field ontologies, Mao et al. (2020) worked on an application-oriented process safety knowledge graph and Gao et al. (2020) on an application-oriented HAZOP ontology.

The complexity of the HAZOP domain has not been considered in the approaches mentioned above. Despite the development of various research approaches and prototypes in the past, the performance of such systems is hardly comparable to the quality of the results produced by experienced HAZOP practitioners.

2. Aim, objectives, and overview

This research aims to digitize and thus automate the HAZOP method. As such, an approach is defined, developed, implemented, and investigated to represent complex interdisciplinary knowledge from the HAZOP domain and utilize it through a knowledge-based system. Using this approach, existing research gaps will be closed, which contributes directly to the general and problem-specific objectives and requirements for this research:

(1) **State of research**: a process involving a creativity method conducted by HAZOP practitioners will be transferred into a virtual workflow. The HAZOP method and procedures of HAZOP practitioners and methodological aspects must be analyzed. In addition, state-of-the-art information technologies for the digitalization of the HAZOP method must be identified and evaluated. The current state of research will be evaluated to gain detailed insights into past research approaches.

(2) **Digital HAZOP**: the research question is how and with which technologies can the HAZOP method be digitized? An approach with a technology readiness level (TRL) 4 is envisaged.

(3) **Causation model**: a suitable causation model of scenarios considering all aspects necessary to identify hazardous events and operability issues within process plants will be developed. Different scenario complexities will be considered, e.g., the propagation of partial scenarios through plant sections.

(4) **Expert knowledge representation**: in the context of knowledge acquisition, all subdomains for capturing the interdisciplinary HAZOP domain knowledge from the fields of process engineering, site engineering, process control, safety engineering, and hazardous substances must be identified. In this context, interfaces between subdomains must also be considered. The entire method development process, including derivation of the thought process and explanations regarding the level of detail, will be presented.

(5) **Process plant representation**: a suitable process plant representation, including modeling functional and structural aspects of equipment and connections to adjacent equipment, will be developed. The required degree of abstraction and detail will be investigated to identify plausible HAZOP scenarios. Due to the diversity of processes and process plants in the chemical industry, equipment will be characterized so that scenarios can be transferred without violating their applicability and validity.

(6) **Substance and site information**: all required substance, ambient, and site information necessary to identify scenarios will be described. This information will be represented in a suitable knowledge representation.

(7) **Risk assessment and safeguards**: appropriate safeguards will be reliably identified based on the identified scenarios. Due to the dependence of the extent and quality of safeguards on the risk of scenarios, performing a risk assessment (based on a risk matrix) of scenarios is necessary for proposing safeguards.

(8) **Inference algorithm**: an inference strategy will be developed, investigated, and implemented within an algorithm that identifies hazards and operability issues based on the causation model, knowledge representation, and process plant representation. The following question will be answered: how can a programmatic procedure emulate the

HAZOP method without producing implausible and redundant results? It should also be possible to identify more complex cross-equipment scenarios.

(9) **Results**: the following criteria will be used to assess the quality of automatically generated HAZOP worksheets:

- accuracy: clear representation of scenarios,
- completeness: all safety-relevant scenarios identified,
- relevance: avoidance of redundancy,
- reliability: reproducibility of the worksheets.

This dissertation is organized into eight chapters. After introducing the research topic and describing the objectives, the theoretical fundamentals are covered in Chapter 3. On the one hand, the HAZOP method is presented and contextualized. On the other hand, knowledge-based systems and reasoning paradigms are outlined to provide a basic understanding for analyzing and evaluating the state of the research regarding the automation of HAZOP studies (cf. Chapter 4). Subsequently, Chapter 5 proposes the developed ontology-based computer system for automatically identifying relevant HAZOP scenarios, conducting risk assessment, and proposing safeguards. This chapter also includes the theoretical foundation's description, such as representing the process plant, causal models, and algorithms for scenario identification and propagation. The developed method is applied within different case studies, and the results are presented in Chapter 6. In Chapter 7, specific insights, findings, limitations, further research ideas, and the significance of this research are discussed. Finally, Chapter 8 summarizes this investigation and its contributions to the field.

3. Fundamentals

The following chapter explains hazard identification methods and the HAZOP method in more detail. Besides, technological aspects regarding knowledge-based systems (KBS) are characterized. The terms data, information, and knowledge are distinguished, graph-theoretical basics are explained, and concepts for knowledge representation, inference, and reasoning methods are described.

3.1 Hazard identification methods

Hazard identification (HAZID) is part of the PHA and aims to identify potential hazardous events associated with processes, transportation, and process plants. The hazard analyses of the process plant are the central part of the safety report. In the CCPS Process Safety Glossary, a hazard is defined as "an inherent chemical or physical characteristic that has the potential for causing damage to people, property, or the environment." There are various methods to systematically identify potential hazards, such as a checklist, what-if, fault tree analysis (FTA), Zurich hazard analysis (ZHA), failure mode and effect analysis (FMEA), and hazard and operability (HAZOP) studies (cf. Hyatt, 2003).

The checklist procedure involves systematically processing questions to conclude potential hazardous events. Usually, these questions are answered with *yes*, *no*, or *n/a*. What-if analyses can be assigned to the brainstorming techniques. First, the process plant is divided into nodes, then failures and hazards are postulated using a 'what if' question while consequences, safeguards, and required actions are recorded (cf. Hyatt, 2003). Fault Tree Analysis (FTA) is a deductive methodology based on logical diagrams representing the contributing causes of a hazardous event. It is used in hazard identification or accident analysis (cf. Mannan, 2012).

The Zurich hazard analysis (ZHA) process is described by Clarke, 2000. It is a brainstorming technique that is performed in a team. The first step in a ZHA is defining the scope of a process, facility, or system. Potentially hazardous scenarios are identified using guidewords that can be looked up from a so-called *tickler list*. The identified scenarios are documented within a hazard catalog. Also, the severity and probability of hazardous scenarios are assessed. Based on that, the risk profile is derived, while risk improvement actions are identified and documented.

The FMEA procedure is described in the IEC 60812. FMEA is a structured bottom-up method carried out by a team to identify system failures (cf. Altabbakh, 2013). In the analysis, systems are described, and potential failures, effects, and causes are analyzed and evaluated. Within the study, possible failure modes are identified, while the causes and effects of the failure modes are brainstormed. Typically, the focus is on component-specific aspects. The results are recorded in tabular form. The HAZOP study procedure is described in the IEC 61882 and Section 3.2. In brief, deviations from nominal functions of study nodes are determined. Afterward, possible causes and consequences of these deviations, safeguards, and required actions are brainstormed and recorded.

Besides these established methods, variations or combinations of different methods were developed. MacGregor developed a method named process flow failure modes (PFFM) (cf. MacGregor, 2013), a structured What-if method that follows the process flow within the process plants. Seligmann et al. blended function and component-driven methods in their approach BLHAZID. In the course of this, he developed the *functional systems framework* for representing structural, functional, and process design-specific aspects (cf. Seligmann et al., 2012).

Figure 2 shows the most frequently used methods for hazard identification within risk analyses in Germany and Switzerland. The HAZOP method is most frequently used for hazard identification. The methods described can be divided into component and function-based methods.

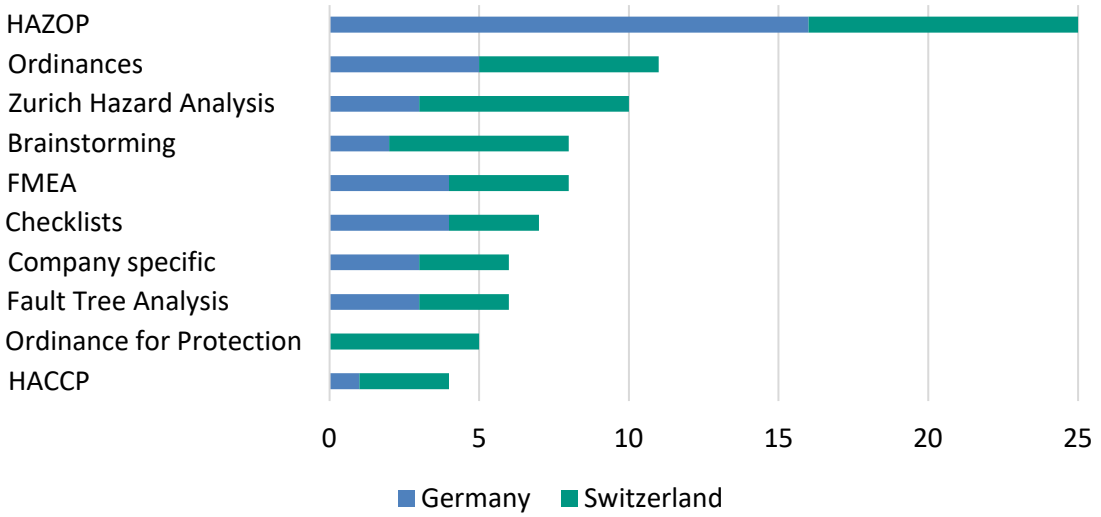


Figure 2: Most frequently used hazard identification methods (cf. Mock and van Mahnen 1998)

The FMEA and FTA method can be assigned to the component-based methods, while Checklist, What-if, HAZOP, and ZHA methods can be assigned to the function or process-based methods. Despite the differences, the methods checklist, what-if, ZHA, FMEA, and HAZOP share multiple aspects, such as:

- it is conducted within a team,
- the results are qualitative,
- an inductive approach is followed (bottom-up),
- typically a single failure is considered as the cause of a hazardous event.

3.2 HAZOP methodology

The HAZOP method is an established qualitative analysis technique within the chemical process industry. It was developed within imperial chemical industries (ICI) in the 1960s in the United Kingdom, and the method was first publicly presented in 1973 (cf. Kletz, 1997). Within a HAZOP study, safety hazards and operability problems within process plants are identified and assessed (cf. CCPS, 2008a). The IEC 61882:2016 defines the HAZOP as a structured and systematic technique for examining a defined system. Its objectives are to identify hazards associated with operating and maintaining a system and identify potential operability problems leading to nonconforming products or loss of production (cf. IEC 61882:2016).

3.2.1 HAZOP terminology

The typical HAZOP terms, their scope, and usage are briefly described below:

- **Study node**: describes the part of the process plant (section of equipment) under consideration. One approach is the subdivision into principal sections, which in themselves perform a process sub-task (cf. Dunjó et al., 2011). Another approach is to distinguish nodes based on their pressure level. Process descriptions and PFDs or P&IDs are needed to divide the process plant into nodes.
- **Design intent**: description of the operation of the process plant or equipment of the node under consideration. In the case of equipment-based HAZOP analyses, the intended function refers to the equipment.

- **Deviation**: a deviation is composed of guidewords and parameters (e.g., variable, activity, component) and describes deviations from the design intent. Various guidewords are used to direct the attention of the HAZOP participants in different directions. The deviation description is occasionally given in text form instead of keywords. For example, "Flow no or less hexane to the tank" instead of 'no/less flow'. The selection of guidewords depends on the type of process plant, process, and operating mode. Not all combinations of guideword and parameter result in meaningful deviations, and accordingly, the physical meaning must be considered.
- **Cause**: describe the reasons for the process deviations under consideration (initiating event). Causes may include operational faults, loss of structural or mechanical integrity, human error, or external events. Several independent causes may lead to the same deviations, which increases their probability of occurrence.
- **Consequence**: describes the results of process deviation. The CCPS (n.d.) Process Safety Glossary defines a *consequence* as "the undesirable result of a loss event, usually measured in health and safety effects, environmental impacts, loss of property, and business interruption costs." In the ISO 61882:2016, which provides guidance for HAZOP studies, a *consequence* is generally described as an "outcome of an event affecting objectives." The objective here refers to the intended function of the node. Consequences include loss events and operability or quality issues. Depending on the underlying philosophy or company guidelines, loss (event) impacts, such as loss of production, explosion, fire, debris, are also considered. Within the industrial practice, consequences usually refer to causes or are a direct consequence of causes. Complete chains of consequences instead of single consequences are described depending on how the studies are conducted.
- **Likelihood**: describes the likelihood of the cause occurring; the description can be qualitative using likelihood categories or quantitative.
- **Severity**: is used to evaluate the extent of loss of the scenario. The severity refers to the consequences of the scenario and is based on different categories, e.g., from minor to catastrophic. The extent of damage depends on the safety of people, damage to property and the environment. Besides, the public image of companies may be considered.
- **Risk**: is determined based on likelihood and severity and is used for risk ranking scenarios. Estimating the risk makes it possible to concentrate on critical scenarios within the study and determine critical safeguards.
- **Safeguard**: the CCPS (n.d.) Process Safety Glossary defines a safeguard as "any device, system, or action that interrupts the chain of events following an initiating event or that mitigates the consequences." In German-speaking countries, preventive safeguards are referred to as primary safeguards (Primäre Schutzmaßnahmen), while mitigative safeguards are referred to as secondary safeguards (Sekundäre Schutzmaßnahmen). Within the German-speaking area, it is common practice not to consider existing safeguards a priori, and they are identified within the scope of the study. The advantage of this is that the danger of a confirmation bias does not exist, and adequate protective measures must always be identified. Internationally, built-in safeguards are not necessarily ignored. The extent of the safeguards is directly dependent on the risk assessment of the scenario.
- **Recommendation/Action:** suggestions for design or procedural changes, (cf. CCPS, 2008b). Recommendations represent a comparison between existing and still missing safeguards to reduce the risk of scenarios to an acceptable level. There are regulations issued by competent authorities that regulate the handling of recommendations. For example, the *Occupational Safety and Health Administration* (OSHA) standard 29 CFR 1910.119 specifies that recorded recommendations are to be resolved and actions taken in a timely manner.

3.2.2 Procedure

The schematic HAZOP process is shown in Figure 3. After initiating the HAZOP study, the first step is to define the scope and objectives, roles, and responsibilities (cf. IEC 61882:2016). Afterward, prerequisite process-safety information, such as the plant's piping and instrumentation diagrams (P&ID) or process flow diagrams (PFD)s, process description, equipment and machines, and site information are gathered. Sometimes, additional documents, such as block diagrams, electrical schematics, mechanical drawings, and pneumatic schematics, are used (cf. Redmill et al., 2000). The prerequisite process-safety information should also include modes of operation, start-up, shut-down, or emergency procedures. When all the prerequisite process-safety information is available, the P&ID and the process description are examined, and the process plant is divided into nodes. These nodes represent sections or lines, which are easier to analyze than the entire process plant. The P&IDs' symbols are interpreted based on professional experience and training, and the process description is used to understand the process. The size of the nodes under consideration is variable, and sometimes only particularly critical sections or interfaces between equipment entities are considered. Afterward, the intended functions of the nodes are examined, defined, and recorded. Potential deviations from the specified intended function are described using guidewords (e.g., more, less, no) and (process) parameters or physical property (e.g., pressure, temperature). Typical combinations of guidewords and parameters are shown in Table 1.

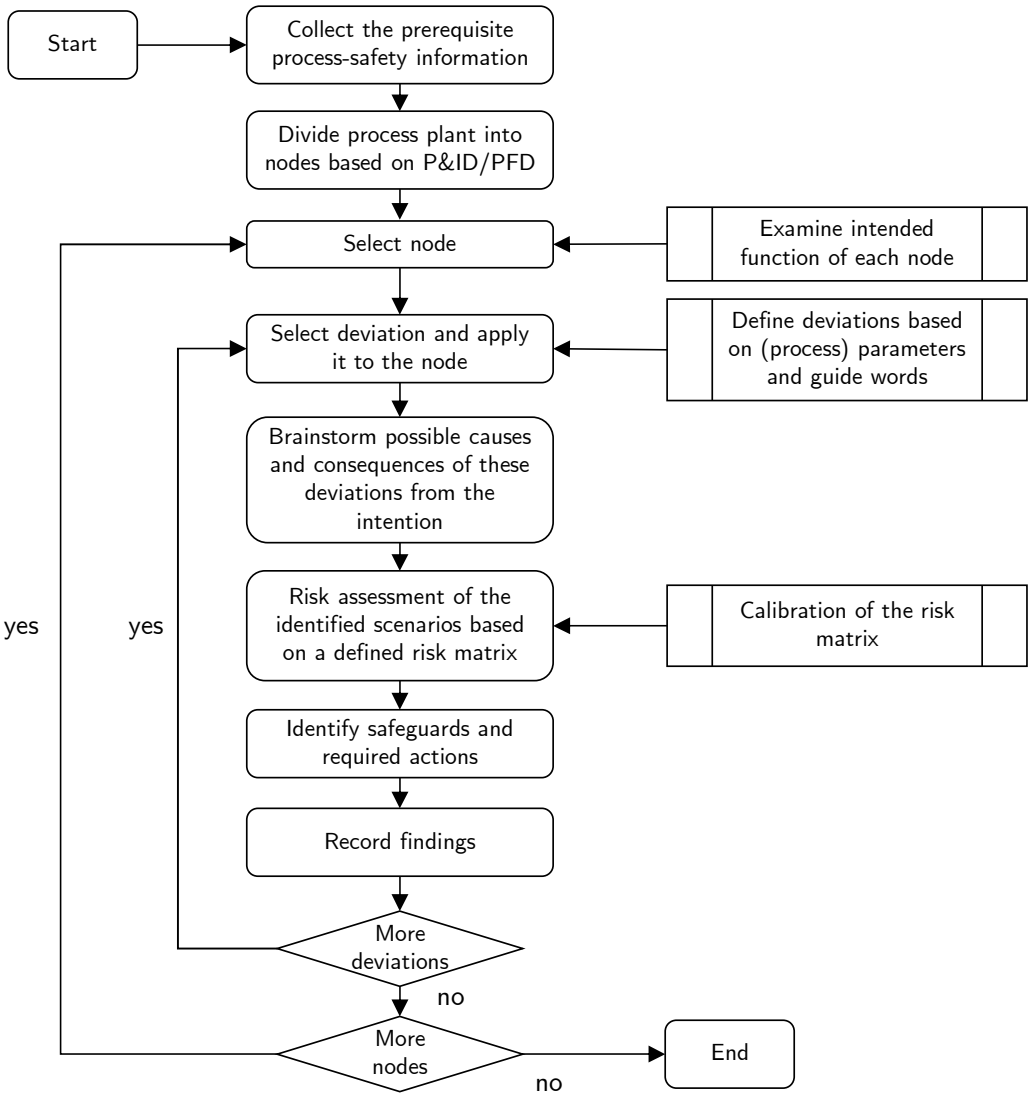


Figure 3: Schematic representation of the HAZOP process

Table 1: Typical guide-word and parameter combinations (cf. Crawley and Tyler, 2015b)

Guideword \ Parameter	No	More	Less	Reverse	Part of	As well as	Elsewhere	Other than
Flow	✓	✓	✓	✓			✓	✓
Temperature		✓	✓					
Pressure	✓	✓	✓					
Composition					✓	✓		
Reaction	✓	✓	✓					✓
Level	✓	✓	✓					

Deviations are used to indicate potential hazardous scenarios, equipment failures, or operability problems and, combined with the nodes, are the starting point of the brainstorming. Site information must also be considered during these considerations. Not always all deviations are applied to all nodes but selected based on experience. Based on the deviations and causes, consequences are identified. Afterward, appropriate safeguards, recommendations, and required actions are discussed and determined. This procedure is repeated for all specified deviations and nodes. The brainstorming aspect is intended to encourage participants to use their imagination and not just commit to design specifications. Within the brainstorming, different opinions, assessments, and experiences of the HAZOP participants are discussed to reach a consensus. The typical result of HAZOP studies are worksheets created by the HAZOP leader or the scribe based on the study's findings. The findings and the documentation are based on consensus, which means the HAZOP participants agree on it.

3.2.3 Model concept of the HAZOP method

Within the classic HAZOP method, possible causes are explored backward, while potential consequences are explored forwards, starting from a deviation of the design intent (cf. Redmill et al., 2000; Figure 4a). Hence, the cause represents the initiating event, while the deviation follows the cause, representing a shift from the design intent (cf. CCPS, 2008c, pp. 20). Based on the deviation and the cause, consequences are identified. HAZOP practitioners assume that consequences refer to causes. Some practitioners even warn against identifying consequences before causes because some consequences can only be identified based on causes; otherwise, critical scenarios could be missed (cf. Baybutt, 2019).

The underlying model concept plays a decisive role in describing the scenarios used to describe incident sequences. An incident can be described as an undesired event that has an adverse impact or leads to a near miss (cf. CCPS, 2008a). An accident, in turn, inevitably involves harmful impacts. In this context, the term *incident sequence* is mentioned, a sequence of events where a hazard, via cause and deviation, is transformed into a loss event, leading to adverse impacts (cf. CCPS, 2008c; Figure 4b). A loss event is a synonym for a hazardous event, such as releasing toxic material or the rupture of a vessel (cf. CCPS, n.d.).

Rasmussen and Svedung (2000) use the term *anatomy of an accident* to describe an accident's course through a sequence (cf. Figure 4c). The sequence of events starts from a root cause, forming a causal chain leading to a critical event and results in an accidental flow with effects on people, the environment, or assets. For example, an incorrectly opened valve may lead to the deviation 'elsewhere flow' of a flammable liquid, resulting in flash evaporation with subsequent formation of an explosive atmosphere, igniting and exploding, causing debris to fly and injuring people. Figure 4 shows which events are considered in the HAZOP study.

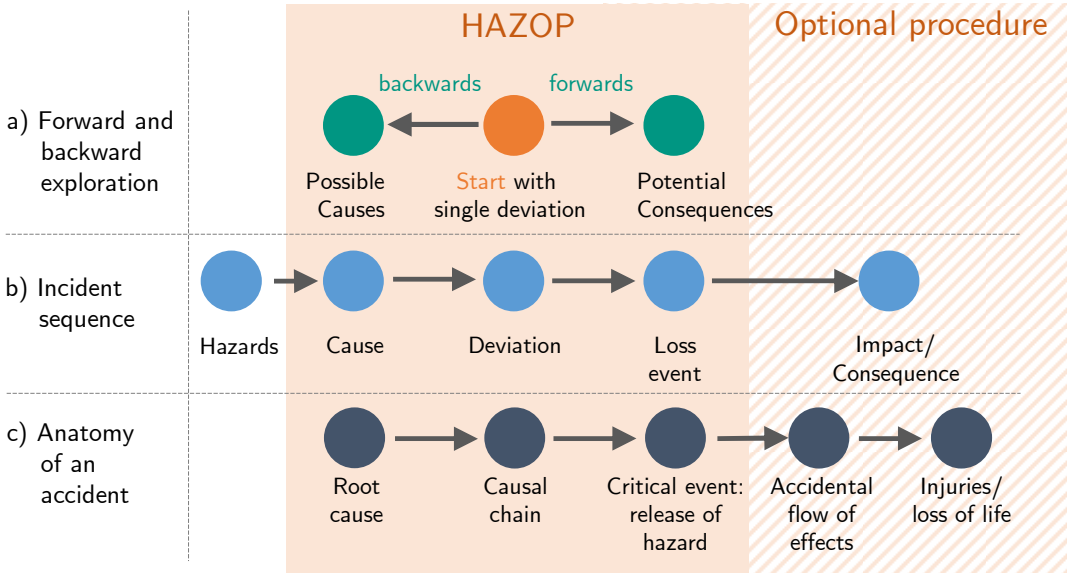


Figure 4: Different model conceptions to describe incidents/accidents

Depending on the approach of HAZOP conductors and company policies, impacts such as loss of life or property damage are not explicitly listed in the consequence column of HAZOP worksheets. During risk assessment, impacts (consequences) are considered to determine appropriate safeguards or further measures. Depending on company-specific guidelines, detailed consequence analyses may already be conducted as part of the HAZOP study or afterward.

3.2.4 Concept map of working concepts

Working terms are provided with relations within a concept map in Figure 5 to classify them in the scope of this work and the conception of the HAZOP model. Concept maps are used to structure and explicitly represent concepts. In the CCPS (n.d.) Process Safety Glossary, a scenario is described as "a detailed description of an unplanned event or incident sequence that results in a loss event and its associated impacts." A scenario can be subdivided into its components, the *scenario components* such as *deviation*, *cause*, and *consequence*. More generally, a *consequence* represents a *hazardous event* synonymous with a *loss event* (cf. CCPS, n.d.). Besides, a *consequence* can also represent an *operability event*, e.g., compressor breakdown. Since a consequence can lead to subsequent consequences, resulting in a *consequence sequence*. The scenario component *cause* is also referred to as *initiating event*, while a *deviation* stands for a *fault event*. The *event sequence* is composed of multiple *events*. The term *event sequence* and *causal chain* may be synonymously used and represents the *causation model*. An *event sequence* can therefore represent a *scenario*.

3.2.5 Team composition and competence

Typically a HAZOP team consists of a project/design engineer, commissioning manager and control/electrical engineer operator, process engineer, process control engineer, research chemist, and safety engineer (cf. Swann and Preston, 1995). Depending on the type of HAZOP, the plant manager also participates in HAZOP studies. Kletz (1999) notes that the HAZOP team's composition can change, for instance, during the design phase of a new process plant and the study of an existing process. A scribe takes over the task to document the results of the brainstorming. Often, the HAZOP study leader also takes over the task of the scribe and documents the study. In this context, Kappelmaier et al. (2008) note that the team must be well balanced in combining knowledge and professional experience. The team must be competent enough to clarify questions independently and not frequently rely on external experts.

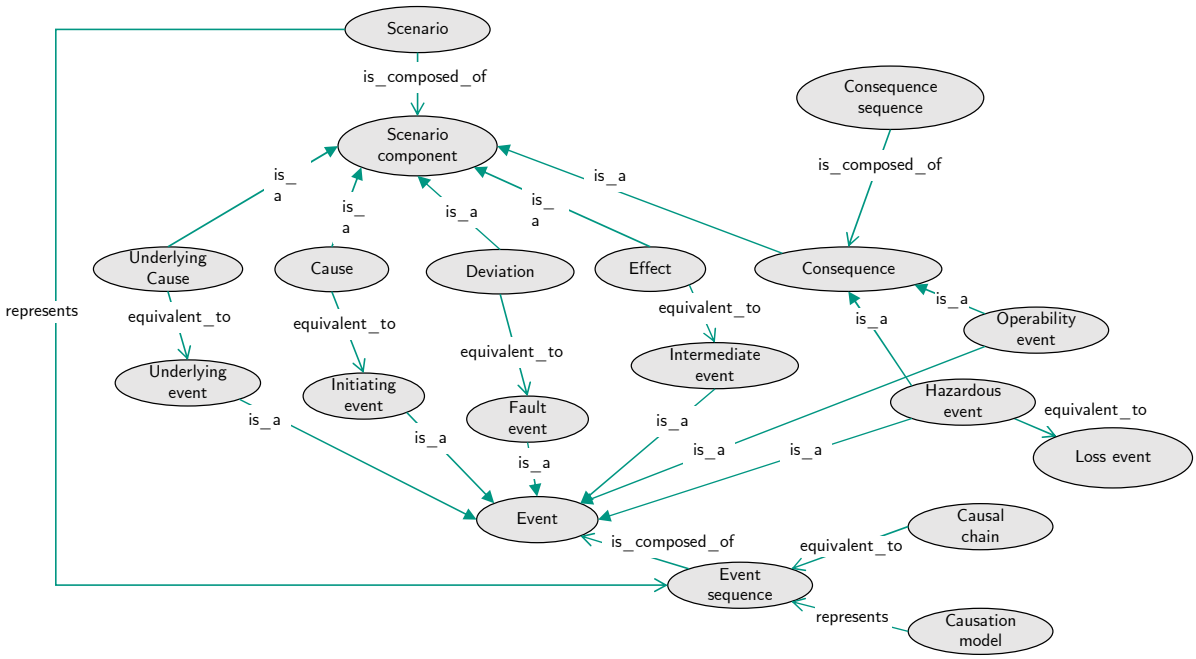


Figure 5: Concept map of working concepts

Also, the team members must be authorized to make decisions in their field (cf. Kappelmaier et al., 2008). Crawley and Tyler (2015a) state that the validity of the HAZOP study depends on the people in the team, the accuracy of the information used (e.g., up-to-date P&ID), and the design quality (e.g., processes designed safe). Swann and Preston (1995) state that HAZOP studies' effectiveness depends on the features: good pre-work, team composition, team leadership, and team leader competence.

3.2.6 Risk assessment as part of the HAZOP study

It is common to combine the HAZOP procedure with risk assessments within the German chemical industry. The application guide of the IEC 61882 standard does not take risk assessment into account. If the risk assessment is included, scenarios are risk ranked, which means causes are assigned a likelihood, while the consequences are assigned a severity (cf. Single et al., 2020a). Based on this, scenarios are ranked according to a risk class. Figure 6 shows an industry-typical risk matrix (cf. Schwarz et al., 2019). In the matrix, a distinction is made between four risk classes:

- A: substantial, unacceptable risk,
- B: considerable, unacceptable risk,
- C: undesirable but tolerable risk,
- D: acceptable risk.

Likelihood	Quantitative	10^{-5} - 10^{-4} p.a.	10^{-4} - 10^{-3} p.a.	10^{-3} - 10^{-2} p.a.	10^{-2} - 10^{-1} p.a.	10^{-1} - 1 p.a.	> 1 p.a.
	Qualitative	**Very unlikely**	**Unlikely**	**Possible**	**Occasional**	**Likely**	**Very Likely**
Severity category	**Catastrophic**	C	B	B	B	A	A
	Severe	C	C	B	B	B	A
	Serious	D	C	C	B	B	B
	Significant	D	D	D	C	C	B
	Minor	D	D	D	D	C	C

Figure 6: Risk matrix based on Schwarz et al. (2019)

The risk classes are composed of the severity and likelihood categories. Qualitative risk categorizations are mainly used in practice, while the matrix is calibrated based on industry-specific requirements. Calibration in this context means that the matrix can be adjusted by allocating the risk classes. A qualitative categorization can also be converted into a quantitative categorization. Within an EPSC working group, risk matrices of renowned companies from the chemical industry were compared within an anonymous survey (cf. Schwarz et al., 2019).

The likelihood classes typically differ by one order of magnitude. In the HAZOP study, an unmitigated risk is first determined without taking safeguards into account. Then, depending on the scenario-specific risk, options for safeguards are proposed that reduce the risk to an acceptable level. Thus, selecting suitable safeguards is directly related to the risk assessment of the scenarios.

3.2.7 Documentation of the study results

It is common to record all the findings in table form in HAZOP worksheets (cf. Table 2). The chosen form of reporting the results may vary in detail, as it depends on company policies, the working methods of consulting engineers, and the requirements of authorities. Necessarily, HAZOP worksheets contain scenario-specific details, such as deviations, causes, consequences, safeguards, and required actions. For instance, some HAZOP worksheet layouts provide a separate column for the node under consideration. Also, worksheets can contain the column *Action recommended/Recommendations*, which describes the action to be taken if existing safeguards are inadequate (cf. Section 3.2.1).

An industry typical HAZOP worksheet is shown in Table 2. It is common to refer to other scenarios when documenting the results within conventional HAZOP studies. Cross-references occur if a consequence implies another process deviation or a scenario shares the same deviation. In Table 2, cross-references are shown in cell 1.3.1 and cell 1.3.1.1.1. The result worksheets are extended by the likelihood, severity, and risk class if the risk assessment is included within the HAZOP study. The likelihood is related to the cause, while the severity is related to the consequence. The selection of safeguards depends on the risk category of a scenario. Thus, safeguards reduce risk by reducing the likelihood of occurrence or mitigating the consequences. The risk ranking usually distinguishes between the unmitigated risk ranking (without safeguards) and the mitigated risk after considering the safeguards. The columns concerning risk assessment are colored in Table 2.

In addition to identifying scenarios, general operational or plant-specific improvements, maintenance, or operational guidance may be additional study outcomes. The documentation must also be complete and comprehensible for persons not involved in the HAZOP study (cf. Kappelmaier et al., 2008). The numerous abbreviations and acronyms used make the worksheets difficult to follow in some cases. Trujillo *et al.* (2015) state that a HAZOP worksheet should reflect the current knowledge of process hazards, consequences, and safeguards. They state that changes in process safety information should lead to a review; otherwise, the HAZOP method loses its effectiveness (cf. Trujillo et al., 2015).

Usually, HAZOP study software is used to document the HAZOP study and to organize the process. (Turney, 1991) notes that documentation software can improve meetings' efficiency, more accurate action lists, actions completed more rapidly, fewer misunderstandings, and better records. Also, the documentation software is used for data storage and organization, i.e., all required documents, such as P&IDs, process descriptions, or datasheets, can be attached to every HAZOP project. Furthermore, participants can be informed, and responsible persons can be specified. A widely used software solution is *PHA-Pro* (cf. Sphera, n.d.) and *PHA Works* (cf. Primatech, n.d.).

Table 2: Industry typical HAZOP worksheet

Node description	Hexane storage tank with inflow line								
Drawings / References	P&ID: D-007;								
Design Intent/Parameters	Storage of liquid hexane, MAWP = 0.5 barg @ 293 C								
Deviation	Causes	Consequences	Consequence Severity	Safeguards	Cause Likelihood	Unmitigated Risk Rankings	Mitigated Likelihood	Mitigated Risk Rankings	Recommendations
1.1 No or less flow to hexane tank	1.1.1 Tank inlet valve closed	1.1.1.1 Level drop of tank 1.1.1.2 Dry running of pump	2	1.1.1.1.1 Dry run protection for pump	3	2			
1.2 High filling level	1.2.1 Control loop failure	1.2.1.1 Overfill; exceeding MAWP; failure Tank	4	1.2.1.1.1 Overfill protection that closes the inlet valve 1.2.1.1.2 Relief valve that discharges to a collectin basin 1.2.1.1.3 Level indication with high level alarm	3	3			
1.3 High pressure	1.3.1 High level (see 1.2)	1.3.1.1 Exceeding MAWP; failure Tank	4	1.3.1.1.1 see 1.2.1.1.3	3	3			

3.2.8 Other applications of the HAZOP method

In computer science, HAZOP studies are also applied to analyze the information flow inside software components instead of a material or energy flow (cf. McDermid et al., 1995). Computer HAZOP (CHAZOP) is a variation of the HAZOP method to investigate computer-based control systems, such as programmable logic controller (PLC), input/output signals, and computer systems (cf. Schubach, 1997). For the analysis of computer-based systems, different documents, such as data flow diagrams, state transition diagrams, or object-oriented diagrams, are required (cf. Redmill et al., 2000).

3.3 Evaluation of the HAZOP method

The advantages and disadvantages of the HAZOP method are examined for valuation purposes in Table 3. One of the significant disadvantages is the time required:

- a typical HAZOP study takes one to eight weeks (cf. Venkatasubramanian et al., 2000),
- each deviation is discussed on average, ten minutes per node (cf. Swann and Preston, 1995),
- each main plant item takes 1.5-3 hours (cf. Kletz, 1999).

These time specifications do not consider the risk assessment part, which is often integrated into the HAZOP study within the industrial practice. Venkatasubramanian et al. (2000) state that 70 % of the time and effort are devoted to analyzing routine process deviations. Khan and Abbasi (1997b) developed a mathematical model to estimate the time spent on a HAZOP study. They note that the HAZOP leader's skill and the number and complexity of the involved P&IDs significantly influence the time required for the study (cf. Khan and Abbasi, 1997c).

In addition to the high time expenditure, there are method-specific disadvantages. The classic HAZOP method is subject to the single cause assumption, which means that causes are considered independently and not in combination. The occurrence of two independent deviations leading to a hazardous scenario is called *double jeopardy*. For example, the failure of an agitator (1st deviation/cause) while adding an excessive amount of catalyst (2nd deviation/cause) to a reactor.

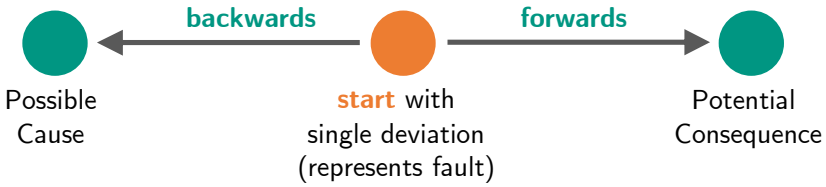


Figure 7: Causality model behind the HAZOP model

For the above-average risk potential of scenarios, double jeopardy is considered within the chemical industry, although double jeopardy is not considered in a standard HAZOP study. Additionally, common cause failures (CCF) are partially considered within HAZOP studies. The CCPS (n.d.) Process Safety Glossary defines common cause failures as "Concurrent failures of different devices, resulting from a single event, where these failures are not consequences of each other." Common cause failures are considered within HAZOP studies by assuming a global loss of services, such as global loss of steam, power, instrument air, or nitrogen (cf. Roche *et al.*, 2019).

When using the HAZOP method, individual deviations are associated with causes and consequences, leading to missed scenarios (cf. Roche et al., 2019). For instance, causes or consequences cannot be matched to specific deviations because the deviations do not fit the scenarios. The original idea behind the HAZOP method is to start the analysis at the middle of the event sequence with the deviation (cf. Figure 5; Figure 7). Thus, causes or consequences that cannot be linked to a specific deviation are not identified.

In case multiple components of the system are highly interlinked and interact strongly, the IEC 61882 proposes to consider event tree analysis (ETA) or fault tree analysis (FTA) in addition to the HAZOP method. Faults are described by deviations and are assumed to occur only one at a time. As shown in Figure 7, the HAZOP method is based on a linear chain of events model. Leveson (2019) criticizes linear models for leaving out the most important causal factors of accidents today. The reason for this is that systemic factors in accidents and indirect or non-linear interactions among events are excluded, such as the effectiveness of process control (cf. Leveson and Stephanopoulos, 2014).

Besides methodological aspects, the team's competence has the most significant influence on achieving the quality of the results. Hence, the multidisciplinary team must have an in-depth process understanding. More general, a HAZOP team should have technical competencies, such as education, professional experience, skills, training, and personal competencies, such as behavior and attitudes. Human misconception and cognitive bias decrease the quality of the achievable results of HAZOP studies.

Hofinger (2008) names the following causes of human errors: excessive demand (external requirements), overfatigue, environmental influences (noise), and unclear goals. Also, the studies are exposed to cognitive distortions, such as *confirmation bias*. It refers to the tendency to confirm assumptions once made by a one-sided search for information and perception of verified facts (cf. Schaub, 2008). Trammell and Davis (2001) described a case during a component-by-component review of an FMEA; the team became frustrated because most components assessed had minimal impact on the system. Thus, the team skipped the review of specific components assuming that no hazards existed. Hence, components were reviewed based on personal history or experience. This type of misconduct can also be transferred to HAZOP studies, e.g., for 'standard' utilities. Furthermore, Duhon (2017) points out the following limitations in this context:

- **tunnel vision**, which means that the focus is on the equipment and not on the node in its entirety,
- **dividing HAZOPs along contract boundaries**, e.g., in the case of different involved organizations or consultants.

Recording scenarios in various places can lead to cross-references within HAZOP worksheets. Duhon (2017) uses the term 'guideword overlap', e.g., when pressure, temperature, or level deviations arise from flow deviations leading to cross-references. Repetitions are beneficial because they support the plausibility of the scenario and increase the probability of recognizing them. Nevertheless, they lead to readability issues of the HAZOP report since scenarios are recorded in various places instead of all at once. In addition, frequent repetitions potentially induce inattention of the participants of the studies. Baybutt (2015) described various weaknesses in HAZOP studies, which concern different aspects of the study:

- Depending on the complexity of the process plant, HAZOP studies tend to be incomplete,
- HAZOP studies take more time than other methods,
- The consideration of compound deviations is not common practice (e.g., double jeopardy),
- Equal scenarios could have more than one deviation, but repetitions can frustrate team members and reduce their alertness,
- In case the focus is on individual nodes propagating deviations could be overlooked,
- The focus on deviations may distract from identifying initiating events of scenarios,
- The HAZOP method does not guide the identification of initiating events. Thus, this depends on the engineering judgment of the HAZOP team.
- Operability scenarios usually outnumber hazard scenarios significantly. Hence, they increase the required time for studies significantly.
- The consideration of every conceivable deviation from the design intent should not be confused with the completeness of scenario identification,
- HAZOP studies are not well-suited to update previous studies since the HAZOP study structure inhibits a focus on changes.
- The effects of changes to processes can propagate through the process plant, leading to modifying existing scenarios. It also makes the recording of new scenarios more complex.

Baybutt (2015) addresses that "there are other elements of scenarios that may be important and often are not recorded in HAZOP study worksheets." The author means that *intermediate events* between deviations and consequences are not part of typical HAZOP worksheets. Baybutt (2015) notes that scenarios are more complicated for reviewers to understand if intermediate events are omitted. Figure 8 shows the difference between industrial practice and the idea of Baybutt (2015) by considering an *intermediate event* in the scenario sequence. The deviation follows the cause within the scenario sequence, while in a HAZOP worksheet, causes are identified based on deviations. Therefore the *deviation* occurs twice in Figure 8.

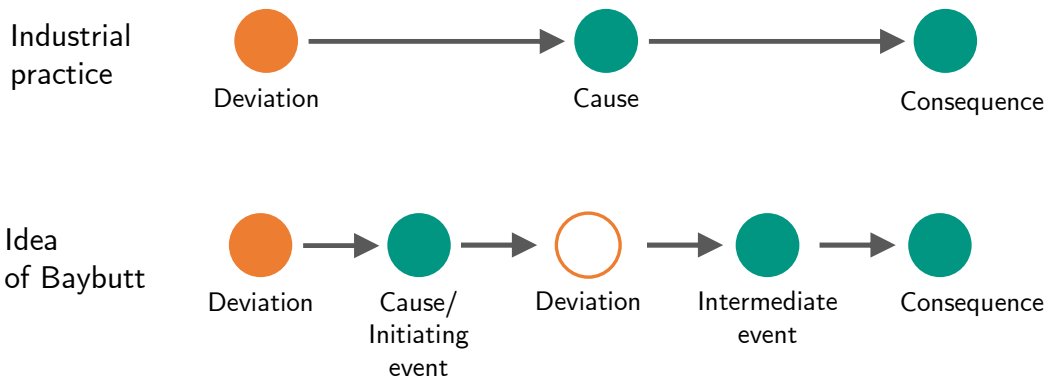


Figure 8: Comparison of industrial practice and intermediate event idea

Table 3: Advantages and disadvantages of HAZOP studies I

Aspect	Advantage	Disadvantage
Methodology	Systematic and rigorous; (Maragakis *et al.*, 2009)	Time-consuming and expensive; (McBride *et al.*, 2009)
	Helps to understand hazards and initiators; (McBride *et al.*, 2009)	Results can vary considerably due to the subjectivity and composition of the HAZOP team; (Seligmann *et al.*, 2012)
	A thorough examination of the equipment; (Kotek and Tabas, 2012)	Lack of representation of human error; (Pasman and Rogers, 2016)
	Line-by-line scrutiny of P&ID; (McBride *et al.*, 2009)	Can inhibit imaginative thinking; (Maragakis *et al.*, 2009)
	Can be applied to operational systems and proposed designs; (Redmill *et al.*, 2000)	Requires a considerable amount of preparation; (Maragakis *et al.*, 2009)
	Particularly effective for novel systems or technologies; (Redmill *et al.*, 2000)	It relies on backward thinking, e.g., hazardous events are preceded by deviations; (MacGregor, 2013)
	Brainstorming method promotes creativity; (own)	HAZOP lacks a systematic approach of elucidate different conclusions into an integrated outcome; (Othman *et al.*, 2016)
	-	Based on a linear chain of events model; (cf. Leveson, 2019)
Scope	Hazard and operability issues are addressed simultaneously; (McBride *et al.*, 2009)	Unclear definition of objectives lead to boundless studies with unclear results; (Kotek and Tabas, 2012)
	A better understanding of the process; (Kotek and Tabas, 2012)	-
	Improve operational regulations; (Kotek and Tabas, 2012)	-
Participants	Views from multidisciplinary experts are considered; (Maragakis *et al.*, 2009)	Requires experienced practitioners; (McBride *et al.*, 2009)
	Team conclusions are more valuable than conclusions of individuals (own)	High demands on the knowledge and skills of participants; (Kotek and Tabas, 2012)
	-	The study depends on the team's harmony and a strong culture of discussion; (own)
	-	The study can rely strongly on the skills of HAZOP leader; (Maragakis *et al.*, 2009)
	-	The team rushes the review or take shortcuts to save time; (MacGregor, 2013)
	-	The consideration of double or multi jeopardy events depends on HAZOP participants (own)
Documentation	Leads to detailed and auditable record; (Maragakis *et al.*, 2009)	Complete recording (all deviations) generate large volumes of paper or large computer files which contain no useful information; (Swann and Preston, 1995)
	Records can be easily updated; (McBride *et al.*, 2009)	Usually requires a considerable number of documents, e.g., P&ID, process description (own)
	-	A large number of cross-references in worksheets

3.4 Overview of graph theory

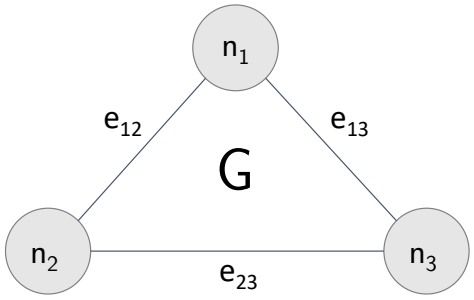


Figure 9: Exemplary simple graph

In this work, graphs play an essential role and are the basis of multiple HAZOP automation approaches from literature and the developed approaches of this work. Graph theory is a field of mathematics that studies graphs. Wilson (1996) describes a simple graph as a finite set of *nodes* and *edges*. In Figure 9, the simple graph G is shown. Graph G whose node set $N(G)$ is $[n_1, n_2, n_3]$ and the set of edges $E(G)$ consists of the edges e_{12}, e_{13}, e_{23}.

Concerning process engineering, graphs can represent equipment and the connections between them or relationships between process variables. There are different types of graphs that are briefly described in the following.

Signed directed graphs (SDG) are graphs with oriented and labeled edges. The edges represent the relationships between nodes, while nodes represent events, situations, objects, concepts, or variables. Hence, equations can be translated and visualized into a graph. Research approaches used SDGs to represent relationships between process variables (cf. Oyeleye and Kramer, 1987). Also, SDGs were used to model cause-effect relationships, where directed edges lead from cause to effect nodes (cf. Venkatasubramanian et al., 2003). Palmer (1999) used SDGs to model the propagation of faults and relationships between deviations, causes, and consequences. Cui et al. (2008) extended the SDG method and proposed the layered digraph model (LDG). Another representative of graphs is *higraphs* (cf. Harel, 1988). They are based on state charts and an extension of regular graphs. Higraphs can be used to develop functional-structural models (cf. Section 4.1.9).

Additionally, hypergraphs generalize graphs, which means that edges are called hyperedges and can connect two or more nodes (cf. Black, 2014). Rossing *et al.* (2010) state that the functional modeling paradigm *multilevel flow models* (MFMs) form hypergraph-like structures. Petri net structures are mathematical representations of systems and can describe causal relationships between events (cf. Peterson, 1981). A Petri net consists of a set of places P, a set of transitions T, an input function I and output function O (cf. Peterson, 1981). Thus, it can be considered a 4-tuple $G = (P, T, I, O)$.

3.5 Data, information, and knowledge

The terms *data*, *information,* and *knowledge* are defined to describe the ideas behind knowledge-based systems. Aamodt and Nygård (1995) describe *data* as patterns without meaning, while *information* represents interpreted data. By adding relationships and providing context, data is given meaning, and *data* becomes *information*. Identifying relationships is essential in developing a knowledge representation and represents an essential aspect of this work (cf. Section 5). Knowledge is regarded as *justified true beliefs* representing learned *information* (cf. Ichikawa and Steup, 2018). Knowledge also includes learned and proven skills. The learning process is described as acquiring new knowledge (cf. Aamodt and Nygård, 1995). In Figure 10, the difference between these terms is shown using an example. The term 'IEC 61882' represents *data*; by adding context, it becomes *information*. Through an awareness of the intention of 'IEC 61882', information becomes *knowledge*. Furthermore, the distinction between different knowledge types is introduced to illustrate the effort involved in knowledge acquisition, modeling, and management. Faust (2007) distinguishes between *explicit*, *tacit,* and *implicit* knowledge.

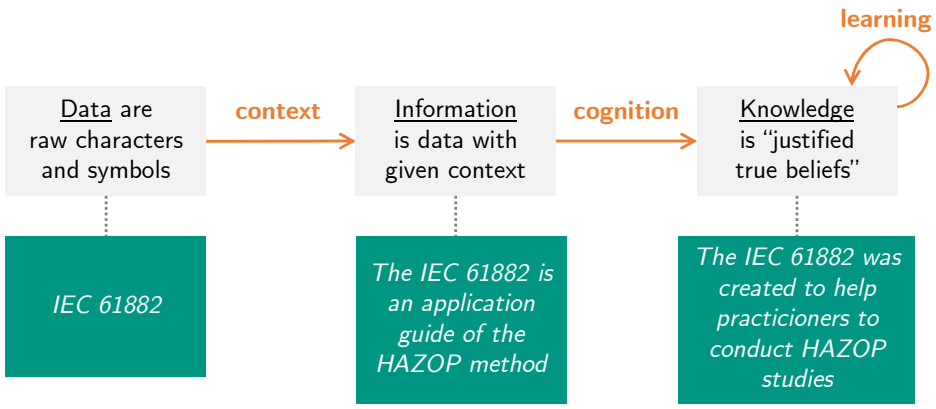


Figure 10: Relationship between data, information and knowledge

Explicit knowledge can be acquired via books, documents, or reports and can be expressed and transmitted in oral or written form and encoded using formal languages. Implicit or tacit knowledge is based on experience, memories, and beliefs that cannot easily be transferred verbally or in writing. In safety engineering, explicit knowledge includes accident records, safety regulations, and guidelines (cf. Hadikusumo and Rowlinson, 2004). Safety hazard recognition can be characterized as tacit knowledge since it depends on the professional experience of the safety engineer (cf. Hadikusumo and Rowlinson, 2004).

Schacter (1994) describes implicit knowledge as "knowledge that is revealed in task performance without any corresponding phenomenal awareness." This definition corresponds to Faust's description, which states that implicit knowledge is difficult to reveal because knowledge carriers (domain experts) cannot describe it because it is too apparent to them (cf. Faust, 2007). This phenomenon was also observed in the course of this research. Especially experts with years of experience are not aware of the cognitive processes involved in solving problems and which experience-based assumptions they automatically make.

Accordingly, the type of knowledge determines how it can be acquired. An attempt should be made to capture and consider experience-based knowledge during knowledge acquisition. HAZOP domain knowledge can be captured through expert interviews, discussions with practitioners, the study of technical literature, and the analysis, evaluation, and interpretation of former HAZOP worksheets.

3.6 From terms, concepts, and things to knowledge models

In the course of modeling knowledge, conceptualization is an important step. Gruber (1993) describes a conceptualization as an abstract, simplified view of the world. In ISO 1087, a concept is defined as a "unit of knowledge created by a unique combination of characteristics." ISO 5963 describes a *concept* as "a unit of thought. The semantic content of a concept can be re-expressed by a combination of other and different concepts, which may vary from one language or culture to another."

A *term* is defined in ISO 1087 as a "designation that represents a general concept by linguistic means." Terms are used to describe *concept*s representing a unit of thought (mental model) to model a *thing* in the real world (e.g., object, phenomena, or circumstance). These relationships can be described through a semiotic triangle (cf. Rousseau et al., 2018). Figure 11 shows a semantic triangle as an example where the term 'fire' invokes the concept of rapid oxidation that releases heat and involves flames which model fire in the real world. Different concepts stand within a context of meaning. In a different context, the meaning of concepts can be different. Thus, concepts can be described by other concepts.

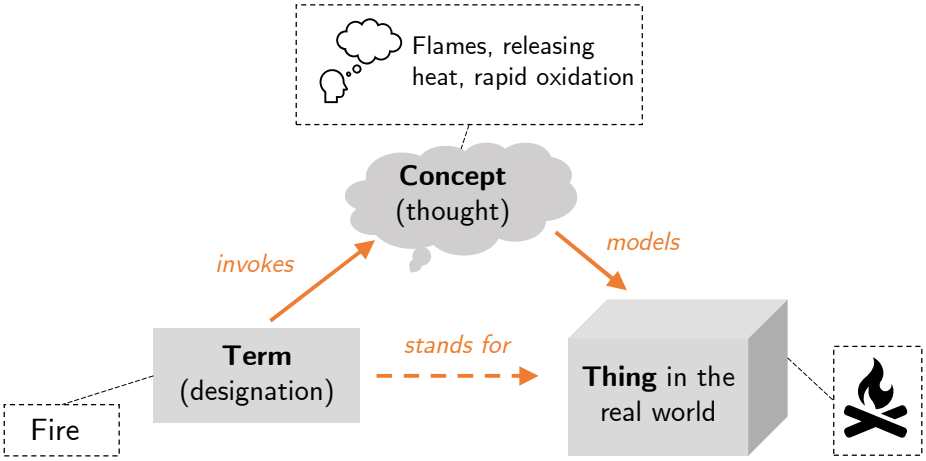


Figure 11: The semiotic triangle applied to a fire

Semantic relationships explain the context of concepts to describe their meaning. A knowledge model represents a domain, application, or task knowledge in a reusable format. From the relationships between concepts emerges the knowledge model that abstracts things in the real world which usually leads to a reduction of reality. There are no direct relations between terms and things within a knowledge representation, but they are described using concepts. This indirect relation is indicated in Figure 11 by the dashed line between *term* and *thing*.

3.7 Knowledge representation formalisms

Knowledge management involves collecting, organizing, and retrieving knowledge. It includes the unambiguous representation of knowledge, making it available to people and computer systems, and finding resources efficiently. An integral part of knowledge-based systems is the knowledge representation (KR) which represents knowledge and facts. Regarding domain knowledge modeling, Noy and McGuinness (n.d.) state, "there is no one correct way to model a domain since there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate."

Davis et al. (1993) describe knowledge representation in terms of five roles (1) a substitute of a thing (surrogate), (2) a set of ontological commitments, (3) a theory of intelligent reasoning, (4) a medium for pragmatically efficient computation, and (5) a medium of human expression. Davis et al. (1993) state that "a knowledge representation is not a data structure". For instance, a semantic net is a KR formalism, while the underlying graph is a data structure (cf. Section 3.6). Hence, a KR is implemented using a particular data structure (cf. Davis et al., 1993). Various knowledge representation formalisms emerged from different disciplines, such as philosophy, artificial intelligence, and cognitive psychology. The KR schemes covered in this research include: *rules*, *frames*, *semantic nets, ontologies* and are highlighted in color in Figure 12.

Knowledge can be expressed using a set of so-called *production rules*. Barr and Feigenbaum (1981) describe a (production) rule as a construct: IF this *condition* holds, THEN this *action* is appropriate. For instance, 'IF flammable substance present, THEN avoid a source of ignition'. These rulesets are stored in rule bases that interpreters evaluate. According to Baldwin and Kasper (1986), "rules represent heuristics for solving a particular class of problems". Even though rule-based systems were invented in the 1940s, they are still recognized knowledge representation formalisms (cf. Barr and Feigenbaum, 1981). Rules are used in logic programming systems, deductive databases, and business rules systems (cf. Grimm et al., 2007). *Semantic networks* (*nets*) express knowledge as linked graphical nodes, ergo graphs (cf. Section 3.4).

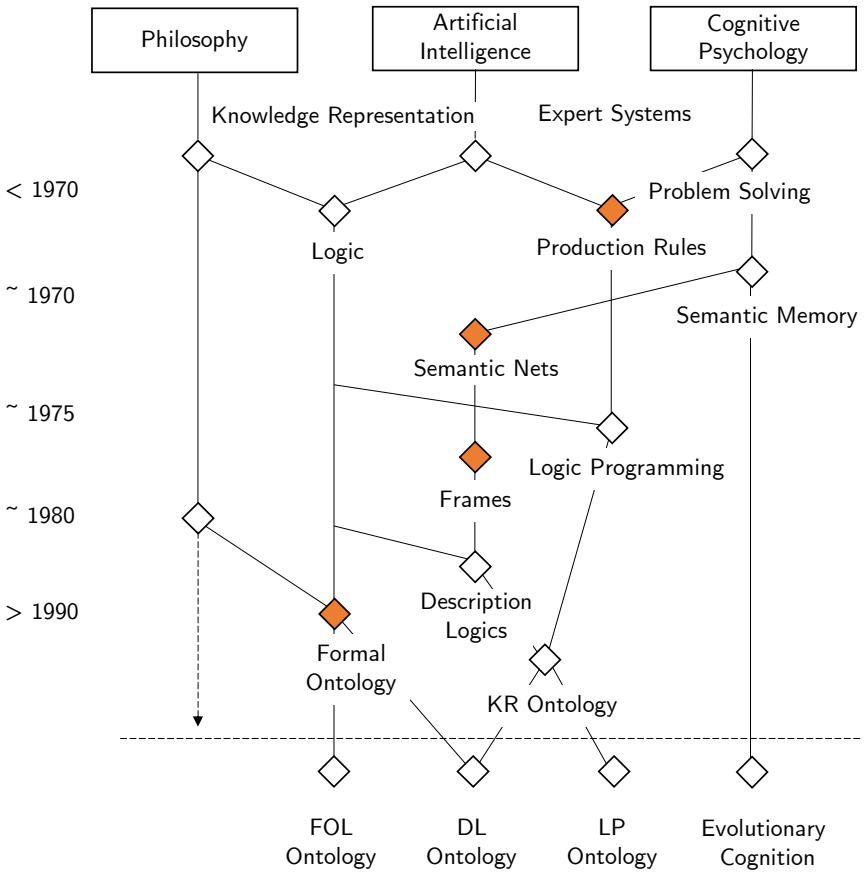


Figure 12: Overview of knowledge representation formalisms (cf. Hoekstra, 2009)

In the late 1960s, researchers developed semantic nets as "psychological models for human memory" (cf. Quillian, 1968; Barr and Feigenbaum, 1981). Within semantic nets, nodes represent concepts, while the edges between the nodes represent relations (cf. Grimm et al., 2007). The idea behind semantic nets is to model knowledge as sets of related (connected) concepts. They can be used to inherit relationships from other nodes, e.g., child nodes can inherit properties from parent nodes. Semantic nets are suitable for expressing concept hierarchies (taxonomies) and general statements about a domain (cf. Grimm et al., 2007).

Minsky (1974) developed data structures for representing stereotyped situations called *frames.* The idea behind it is using "object-oriented groups of procedures to recognize and deal with new situations" (cf. Brachman and Levesque, 2004). Frames are derived from semantic networks and are based on nodes and relations. Furthermore, there are top levels (generic frames) and lower levels (individual frames) consisting of specific slots (properties). The information in the top levels of a frame is always true within a domain. The data in lower levels (slots) can change with new information. Frames contain information regarding facts, data, and procedures. Also, they contain default values regarding data and procedures. Frames and semantic nets are based on formal logic.

3.7.1 Advantages and disadvantages of knowledge representation formalism

Table 4 provides an overview of the advantages and disadvantages of various knowledge representation formalisms. Giarratano and Riley (2005) state that one of the most significant results demonstrated by Newell and Simon (1972) was that much of human problem solving or cognition could be expressed by production rules. This statement may be valid for science and engineering since cause and effect usually depend on each other. Human behavior is an exception to this rule because humans regularly act irrationally, and causal relationships cannot always be explicitly

described. Also, rules are not always suitable in case of incomplete knowledge. Rules cannot always represent implicit knowledge (cf. Section 3.4). Despite these limitations, a rule-based KR has clear advantages. For instance, derived conclusions can be appropriately explained, and general knowledge can be represented compactly (cf. Prentzas and Hatzilygeroudis, 2007). Thus, most past and contemporary approaches use rules (cf. Single et al., 2019a).

Table 4: Advantages and disadvantages of knowledge representation formalisms

Aspect	Advantage	Disadvantage
(Production) **Rules**	Rules are transparent and easy to understand for humans; (cf. Nalepa, 2017)	Complex problems require hundreds or thousands of rules (nested rules); thus new rules cannot be added or removed regardless; (cf. Brachman and Levesque, 2004)
	Rules are intuitive; (cf. Nalepa, 2017)	Inefficient in large domains, (cf. Baldwin and Kasper, 1986)
	Can capture unrestricted logical implications; (cf. Davis *et al.*, 1993)	Below average search efficiency; (cf. Baldwin and Kasper, 1986)
	A uniform structure of knowledge in rule base; (cf. Barr and Feigenbaum, 1981)	Keeping track of arbitrarily interrelated rules is hard; (cf. Lehmann, 1992)
	Modularity: new rules can be added, or old rules removed incrementally (independent of each other); (cf. Brachman and Levesque, 2004)	Computational costs of an exhaustive search through a large space of unconstrained rules can be overwhelming, (cf. Lehmann, 1992)
Semantic nets	Ability to inherit relationships from other nodes; (cf. Bingi, 1995)	Lack formal semantics; (cf. Barr and Feigenbaum, 1981)
	Ability to represent deep knowledge; (cf. Baldwin and Kasper, 1986)	Lack of uniform logical representation; (cf. Nalepa, 2017)
	-	Restricted expressiveness; (cf. Bingi, 1995)
	-	Negation or disjunction cannot be expressed easily; (cf. Baldwin and Kasper, 1986) and (cf. Lehmann, 1992)
	-	Difficult to apply to domains where procedural knowledge is; (cf. Baldwin and Kasper, 1986)
	-	Not (really) suitable to represent concrete individuals or data values; (cf. Grimm *et al.*, 2007)
Frames	Suitable for visual representation, which makes them transparent and intuitive; (cf. Nalepa, 2017)	Lack of explicit logic-based formalization; (cf. Nalepa, 2017)
	Appropriate to model taxonomic relationships between classes; (Davis *et al.*, 1993)	Limited ability to express incomplete knowledge; (cf. Bingi, 1995)
	Probabilistic reasoning, guessing, and default reasoning can be supported; (cf. Baldwin and Kasper, 1986)	-
	Deep knowledge can be represented, (cf. Baldwin and Kasper, 1986)	-
Formal Ontologies	A precise specification of domain model based on formal semantics, (cf. Grimm *et al.*, 2007)	Disadvantages depend strongly on specific languages
	Further advantages depend strongly on specific languages, cf. Section 3.8.3	-

Semantic networks are one of the first formalisms to represent semantic relationships. They can be used to represent deep knowledge, and relationships can be inherited (cf. Table 4). The drawbacks of semantic nets are their lack of formal semantics and expressiveness. Semantic networks can only be used to a limited extent to represent procedural knowledge and the accurate representation of individuals (instances) and data values (cf. Table 4).

Frames are based on semantic nets, which means concepts can be related to each other, and inheritance is possible. Frames can be used to represent general and specific concepts. Additionally, concepts can be provided with attributes and restrictions. Frames are useful to represent stereotyped situations. Fikes and Kehler (1985) describe these situations as "capture the way experts typically think." One drawback is their limited ability to express incomplete knowledge. Further drawbacks depend strongly on the specific structure or language of the frame-based knowledge representation. Table 4 shows further advantages and disadvantages. Since frames are the more expressive successor of semantic networks, semantic networks are not further considered within this work.

3.8 Ontologies

A contemporary knowledge representation formalism is *ontologies* (cf. Figure 12). Peng et al. (2018) state that ontologies can be understood as *directed acyclic graphs* (DAG), where nodes represent concepts and edges represent the relationships between concepts. According to Gruber (1993), an ontology is an "explicit specification of a conceptualization." Here conceptualization refers to the identification of domain knowledge, required concepts, and their relationships. Krötzsch et al. (2014) note that, unlike databases, ontologies do not fully describe a particular situation but consist of a set of statements that apply to the described situation. Thus, an ontology contains concepts that are restricted to constrain their interpretation. During conceptualization, implicit knowledge is usually made explicit and requires abstraction, simplification, and translation of captured knowledge concepts. Gruber (1995) formulates multiple design criteria for ontology development:

- *Clarity*: the meaning of the concepts used in the ontology must be clearly defined,
- *Coherence*: inferences based on logical axioms should be consistent with informal definitions,
- *Extendibility*: new concepts must be addable based on the existing vocabulary,
- *Minimal encoding bias*: the impact of the formalization language on the ontology should be minimized (cf. Schmalenbach, 2013),
- *Minimal ontological commitment* means that as few claims as possible within an ontology should be employed within the covered knowledge domain.

There are different types of ontologies, namely *top-level* (upper), *domain*, *task*, and *application* ontologies (cf. Guarino, 1998). Top-level ontologies describe general concepts common across various domains and independent of specific domains. Domain or task ontologies contain fundamental concepts from a general domain or task. Finally, application ontologies are used for specific application use cases or within a specific domain. Since ontologies are usually created iteratively, extended, and adapted during their life cycle, they are not static. Sometimes a distinction is made between ontologies and knowledge graphs, where collections of domain-specific class structures and their semantic relations are considered ontologies. In contrast, instance-based real-world data in large quantities are considered knowledge graphs. In other words, ontologies are typically schema-oriented, focused on representing concepts, their relations, and axioms. Knowledge graphs are data-oriented, which means they typically contain specific facts based on the schemata of the ontology. In this work, no distinction is made between knowledge graph and ontology, and the term ontology is used for schema-oriented and data-oriented structures. The ontology life cycle

describes the process from the development to the use of the ontology. Fernández *et al.* (1997) name the following stages within the ontology life cycle: specification, conceptualization, formalization, integration, implementation, and maintenance. Ashraf et al. (2015) divide the ontology life cycle into different phases:

- Ontology engineering: designing and developing the ontology,
- Ontology evaluation: measuring the quality of the ontology and making sure the objectives are met,
- Ontology population: populating the ontology with data and deploying it,
- Ontology evolution: Adapting it to new knowledge; preserving consistency; conducting changes regarding the domain, conceptualization, and formal specifications,
- Ontology usage: integrating it in processes; providing feedback; encouraging adoption and re-usage.

The diversity of ontology-based knowledge representations makes it difficult to generalize advantages and disadvantages. Also, there are frame-based ontologies. Thus, fundamental differences between frames and ontologies can only be presented by considering language-specific characteristics. Besides, there are ontology languages that involve rule-based elements (cf. SWRL in Section 3.8.1). Different ontology and frame languages must be compared to conclude which formalism is especially suitable for this research work (cf. Section 3.8.3).

3.8.1 Ontology languages

An ontology must be formalized and implemented using a mathematical or logical description of concepts using a specific language to make an ontology machine-processable. (Stuckenschmidt, 2008) describes the formalization process as the most elaborate and complicated step in creating an ontology. Ontology languages offer particular language constructs for representing terminologies and conceptual structures (freely translated, according to Stuckenschmidt (2008), pp. 157). Also, formal languages ensure the internal consistency of concept definitions (cf. Hoekstra, 2009). According to their structure, various ontological languages can be categorized as *logic-based*, *frame-based*, and *web-based* (cf. Ding, 2001). Ontology languages can also be classified according to the underlying logic type: frame-based, first-order-logic-based, and description-logic-based.

First-order logic (FOL) is a knowledge representation formalism for representing declarative knowledge (cf. Lifschitz et al., 2008). Furthermore, description logics (DL) provide a logical formalism for ontologies. DL is equipped with formal semantics, which allows the unambiguous exchange of knowledge between humans and computer systems (cf. Krötzsch et al., 2014). Most DL are fragments of FOL (cf. Rudolph, 2011, pp. 49). A logical deduction can be used to infer information from the facts that are stated in an ontology. Baader et al. (2008) note that DL-based systems provide reasoning methods to deduce implicit knowledge from explicit knowledge representations.

A classification of established ontology languages is shown in Figure 14. It also shows the development of various ontology languages from the 1970s until today. On the one hand, Figure 14 distinguishes between *traditional* and *web-based* ontology languages, and on the other hand, which paradigm underlies them. One of the first description logic-based ontology languages was KL-ONE (cf. Brachman and Schmolze, 1985). Within KL-ONE, DL is integrated with frame-based knowledge representation (cf. Brachman and Schmolze, 1985). Another traditional ontology language is the *Knowledge Interchange Format* (KIF), which is FOL-based (cf. Genesereth and Fikes, 1992). The F-Logic language is frame-based and used to represent knowledge domains in an object-oriented manner (cf. Angele and Lausen, 2004; Section 3.6). F-Logic has formal semantics (cf. Kifer and Lausen, 1989). In ISO 24707, common logic (CL) is described as a FOL framework intended to exchange and transmit information. The ISO 24707 standard describes the model theory and abstract syntax behind CL.

Figure 13: Example of a triple

The resource description framework (RDF) language is used to exchange machine-understandable information on the Web (cf. Lassila and Swick, 1997). RDF describes web resources and their relationships in a labeled directed graph (cf. Horrocks, 2008). Furthermore, RDF is based on *triples*, consisting of a *subject*, *predicate*, and *object*. Figure 13 shows a triple consisting of the subject 'HAZOP LEADER', the predicate 'LEADS', and the object 'HAZOP STUDY'. Triples form the basis for representing concepts and their relationships. The RDF language can be used to describe classification hierarchies and their relationships. RDF models are expressed using the Extensible Markup Language (XML). RDF schema (RDFS) is also called RDF Vocabulary Description Language (cf. McBride, 2004) and is an extension of RDF. Hitzler *et al.* (2008) describe RDFS as a description language for lightweight ontologies. The ontology language DAML+OIL is a semantic markup language for Web resources (cf. Connolly et al., 2001). It is based on XML and RDF, while the formal rigor of description logic is added (cf. Kalibatiene and Vasilecas, 2011).

3.8.2 Web ontology language (OWL)

The web ontology language (OWL) is an extension of RDF Schema and has its roots in DAML+OIL. OWL is the standard modeling language for ontologies recommended by the World Wide Web Consortium (W3C) in 2004 (cf. Hitzler et al., 2010). Sowa (n.d.) states that the semantics behind RDF and OWL is a subset of CL semantics. A concept map of the essential elements of the OWL language is shown in Figure 15. Within the terminology of the ontology language OWL, concepts are represented by classes, while instances of these classes are called individuals. Classes are related to each other using binary associations called properties (relationships).

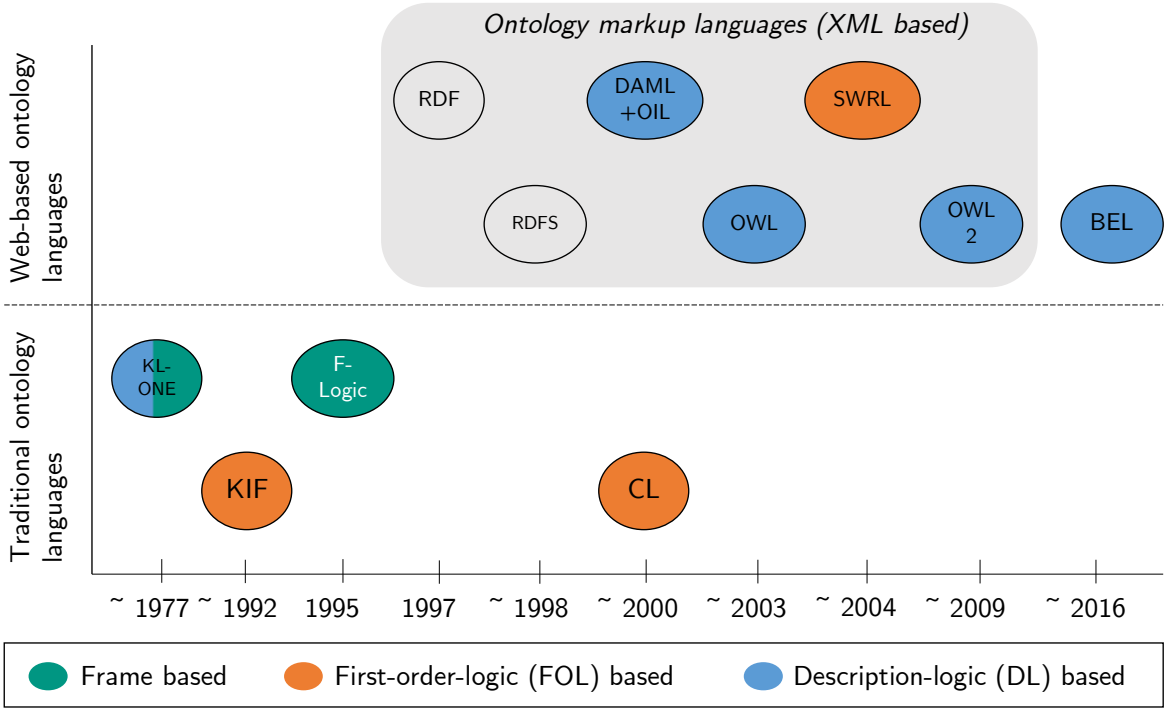


Figure 14: Classification of relevant ontology languages

Axioms describe relationships between ontology elements, such as properties, classes, and individuals (cf. Vrandecic *et al.*, 2006). Axioms are used to constrain the possibilities of interpretations of the classes and relationships in the ontology (cf. Batres and Akmal, 2013). Typical axioms model disjoint, subclass, and equivalent relationships (cf. Figure 15). Additionally, annotations (annotation properties) can annotate ontology elements with metadata, such as labels, comments, and version information. Humans abstract and concretize the meaning of concepts in their minds and put them into context, while within ontologies, context and information are separated (cf. Schmalenbach, 2013). Within DL ontologies, this separation is made with the help of the assertional (ABox), terminological (TBox), and relational (RBox) axioms (cf. Krötzsch et al., 2014). ABox axioms capture knowledge about specific instances (individuals) of the domain, TBox axioms capture knowledge about the concepts (classes) of the domain, and RBox axioms capture knowledge about the relationships (properties) of the domain.

There are three variants (sublanguages) of OWL with different expressiveness levels: OWL Lite, OWL DL, and OWL full. OWL lite can be used to describe classification hierarchies and simple constraints. OWL DL is based on description logics (DL) and represents a compromise between expressiveness and efficient reasoning (cf. Horrocks, 2005). OWL Full supports maximum expressiveness with no computational guarantees (cf. Antoniou *et al.*, 2005). In this context, it is referred to that OWL full is undecidable, which means no currently available software tool supports the entire semantics of OWL full (cf. Hitzler et al., 2010, pp. 138). The semantic web rule language (SWRL) is an extension of OWL (a combination of OWL DL and OWL Lite) with horn-like rules in the form of the rule markup language (RuleML) (cf. Horrocks et al., 2004). OWL was developed in the variants OWL 2 DL, OWL 2 QL, OWL 2 EL, and OWL 2 RL to overcome drawbacks of OWL 1, such as the limited expressivity regarding properties (cf. Grau et al., 2008).

One of the latest developments is the Bayesian ontology language (BEL), an extension of DL EL and can integrate uncertainty with contextual knowledge (cf. Ceylan and Peñaloza, 2017). In this context, the research team around BEL developed a reasoner to process OWL 2 EL ontologies (cf. Ceylan et al., 2015). To the best of current knowledge, it has not yet been implemented within any application. Thus, this ontology language does not appear to be production-ready.

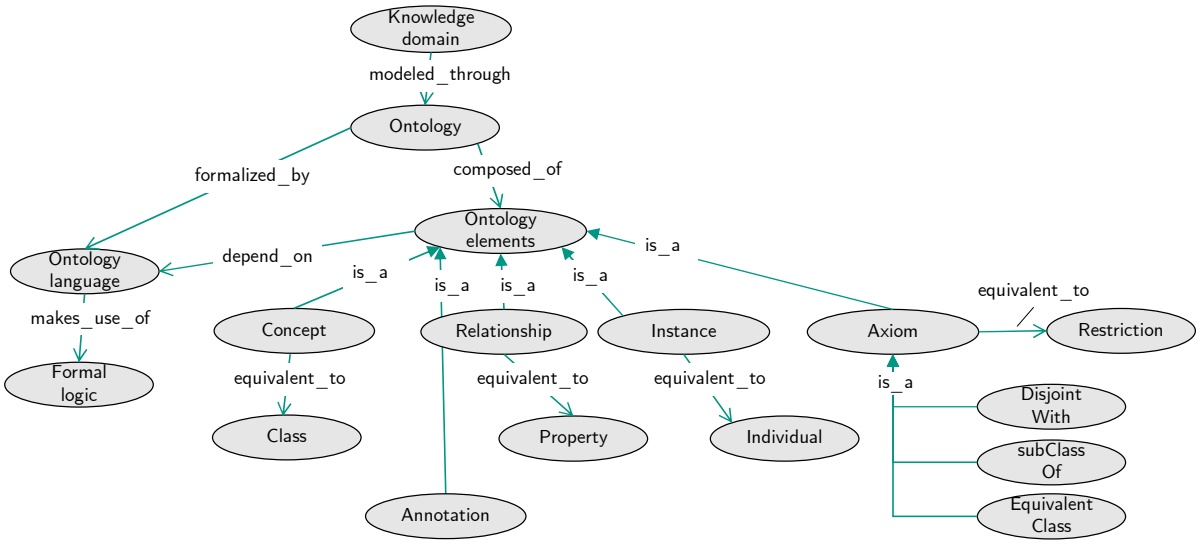


Figure 15: Essential elements of the OWL ontology language

3.8.3 The logical foundation of selected ontology languages

Description logic (DL) belongs to the logic-based formalisms and is used for knowledge representation. The OWL (DL) ontology language is based on description logic. In Table 5, OWL constructors are compared to the DL and Manchester OWL syntax for illustrative purposes (cf. Harmelen et al., 2007; Horridge et al., 2006). Only concepts necessary for clarification are presented. The first example shows an intersection (⊓) between classes A and B, which means the intersection contains all individuals that occur in class A and B (cf. Table 5). In the second example, a class restriction is shown, including the object property r and the class C. Furthermore, it is an existential restriction (∃), which describes individuals with at least one specific property with individuals of a specific class (cf. Horridge, 2011). The third example shows the usage of data types. It consists of an existential restriction (∃), the data property v, and the datatype K. Hence, class restrictions can be formulated as literals. OWL 2 DL has built-in data types, such as strings or floats (cf. Hitzler et al., 2012). In the fourth example, the equivalent class (≡) axiom is shown. Thus, the classes D and E contain exactly the same individuals (cf. Bechhofer et al., 2004).

Table 5: Relationship between OWL constructors, DL and Manchester OWL syntax

	OWL constructor	DL syntax	Manchester OWL syntax	Example
1	intersectionOf	A ⊓ B	A and B	PHA ⊓ HAZOP
2	someValuesFrom	$\exists r.C$	r some C	∃isEffectOfDeviation.LowTemperature
3	someValuesFrom	$\exists v.K$	v some K	∃substanceHasFlashpointInKelvin.<=328.15
4	equivalentClass	D ≡ E	D EquivalentTo E	Fracture ≡ Break

3.8.4 Comparing ontology languages

The choice of ontology language has a decisive influence on the ontology modeling style and determines the reasoners used. Language-specific details are presented below to assess the suitability of ontology languages for this research. Based on the objectives of this work, the general requirements for an ontology language can be stated as follows:

- Recognized and generally accepted knowledge representation language based on state of the art,
- It must be possible to formulate constraints for concepts and relationships to represent complex assertions,
- Datatypes, such as floats, must be representable (e.g., ambient temperature, substance properties),
- Cross ontology inference and queries must be supported,
- Efficient reasoners, editors, evaluation tools, and programming libraries must be available for using the ontology language.

Based on these criteria, available ontology languages are evaluated in the following. The KIF ontology language features full semantic expressiveness, but knowledge representations tend to be larger and less efficient than more restricted languages (cf. Marchetti et al., 2008; Kalibatiene and Vasilecas, 2011). RDF and RDFS have the disadvantage of lacking an effective inference mechanism and formal underlying model semantics (cf. Kalibatiene and Vasilecas, 2011). Compared to RDF, OWL allows the modeling of additional restrictions, such as disjunction of classes, the cardinality of relationships, and property constraints, such as transitivity, uniqueness, or inversion (cf. Schmalenbach, 2013). For instance, in Figure 16, the disjointness between the classes **Deviation**, **Cause**, and **Consequence** means the classes cannot share any instances. The disjointness

axiom allows ambiguous terms to be avoided. Disjointness cannot be expressed using RDFS. The fundamental differences and similarities between the ontology modeling paradigms frames and OWL ontologies are briefly described below.

Frames are based on the *closed world assumption*. According to Wang *et al.*, 2006, this means "everything is prohibited until it is permitted." OWL is based on the *open-world assumption*, which means "everything is permitted until it is prohibited" (cf. Wang et al., 2006). The open-world assumption ensures extensibility. Frames represent typical situations invoked in case of partial or incomplete knowledge and help manage exceptions. Hence, default reasoning is a crucial strength of frames (cf. Wang et al., 2006). On the contrary, OWL "leaves no room for exceptions" (cf. Wang et al., 2006).

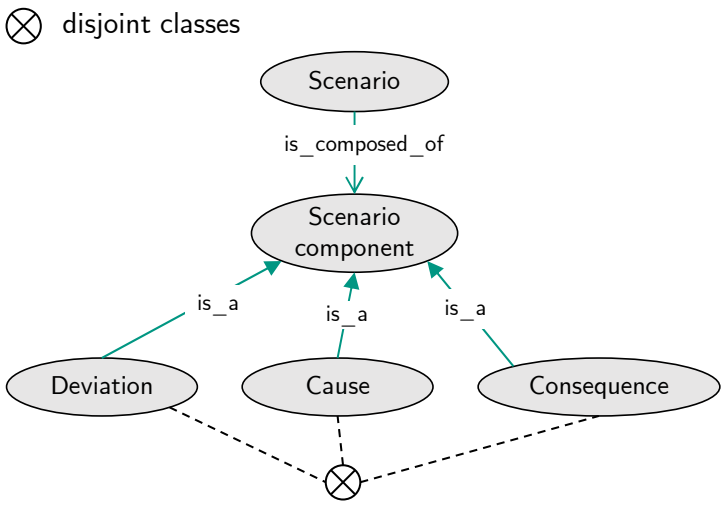


Figure 16: Disjointness between classes

The frame-based ontology language F-logic and systems based on F-logic have recently received little support from the knowledge engineering community. However, commercial systems such as ErgoAI or OntoBroker have their foundation in F-logic (cf. ErgoAI, n.d.; Semafora, n.d.).

Cardoso (2007) investigated the popularity and use of different ontology languages based on 627 surveys collected between 2006 and 2007. OWL and RDFS were the most widely used formal languages (cf. Figure 17). Figure 18 illustrates the popularity of different ontology languages using the number of hits on Google Scholar. The RDFS and the OWL DL language have by far the most hits. Flora-2 is an extension of F-Logic. The popularity of F-Logic decreased within the period under consideration, while the popularity of the SWRL language increased (cf. Figure 18). The queries were performed on 28.11.2020. The following keywords are used to obtain the hits on Google Scholar:

- "F-Logic frame language",
- "Flora-2 frame language",
- "RDFS ontology language",
- "OWL DL ontology language",
- "SWRL ontology language".

The OWL ontology language offers the possibility to provide classes and properties with axioms. For example, different restrictions can be placed on object properties, such as object property axioms (cf. Bock et al., 2012). According to Zhai *et al.* (2018), OWL has expressivity limitations regarding the representation of general rules. For instance, *if-then-else* relationships cannot be represented using OWL 1 (cf. Zhai et al., 2018).

SWRL has strong rule representation capabilities (cf. Zhai et al., 2018). It can be used for statements that cannot be expressed using OWL. Wang *et al.* (2006) state that OWL 1 has difficulties representing quantitative relationships using numeric expressions. This issue can be addressed using SWRL. Also, SWRL rules can be added to OWL ontologies, and therefore no new ontology must be built. Similar to OWL full (cf. Section 3.8.1), SWRL is undecidable, which means that reasoners cannot always perform complete reasoning (cf. Sirin et al., 2007). As part of the advancement of OWL 1, OWL 2 was published in 2009 (cf. Bock et al., 2009). It extends

OWL 1 by features, such as cardinality constructors, extended datatypes, annotations, and increases language expressivity (cf. Grau et al., 2008). Some SWRL rules can also be expressed within OWL 2 (cf. Hitzler et al., 2009). Also, numeric expressions can be expressed using OWL 2. A comparison of the capability of RDFS, OWL 1, and OWL 2 is shown in Table 6. The W3C consortium has declared OWL the standard modeling language (cf. Hitzler et al., 2010). Additionally, the aspects

- usage of OWL 1 and OWL 2 in various current research approaches,
- having a large community (cf. Kalibatiene and Vasilecas, 2011),
- recommendation of the W3C as a standard language,

supports the conclusion that OWL is a recognized and established ontology language. Software libraries are available for working with OWL-based ontologies, such as *Jena*, *owlcpp*, and *Owlready2* (cf. Carroll et al., 2004; Levin et al., 2015; Lamy, 2017). Additionally, the widely used ontology editor with OWL language support Protégé is available (cf. Musen, 2015). Regardless of the underlying knowledge representation paradigm and language, the more diverse and complex the language-specific constructs, the greater the maintenance effort and the resources required for extending the knowledge concepts. Thus, one ontology language should not be chosen because it is more expressive when expressiveness is not required to fulfill the knowledge representation requirements. In summary, KIF and RDFS can be excluded due to inefficient reasoning or the missing ability to restrict relationships and classes. Compared to RDFS, OWL is more expressive (cf. Table 6). Since 2001, DAML+OIL has not been further developed, and OWL is its successor. DL-based languages (e.g., OWL 2 DL) are descendants of KL-ONE.

OWL 2 covers the requirements for an ontology language set at the beginning of Section 3.8.4, such as: recommended by W3C, a large community, stronger expressiveness than RDFS and OWL 1, and numeric relationships can be expressed. Additionally, class definitions and relations can be annotated to document the meaning and context of concepts. Hence, the documentation is part of the ontology, which can even be queried.

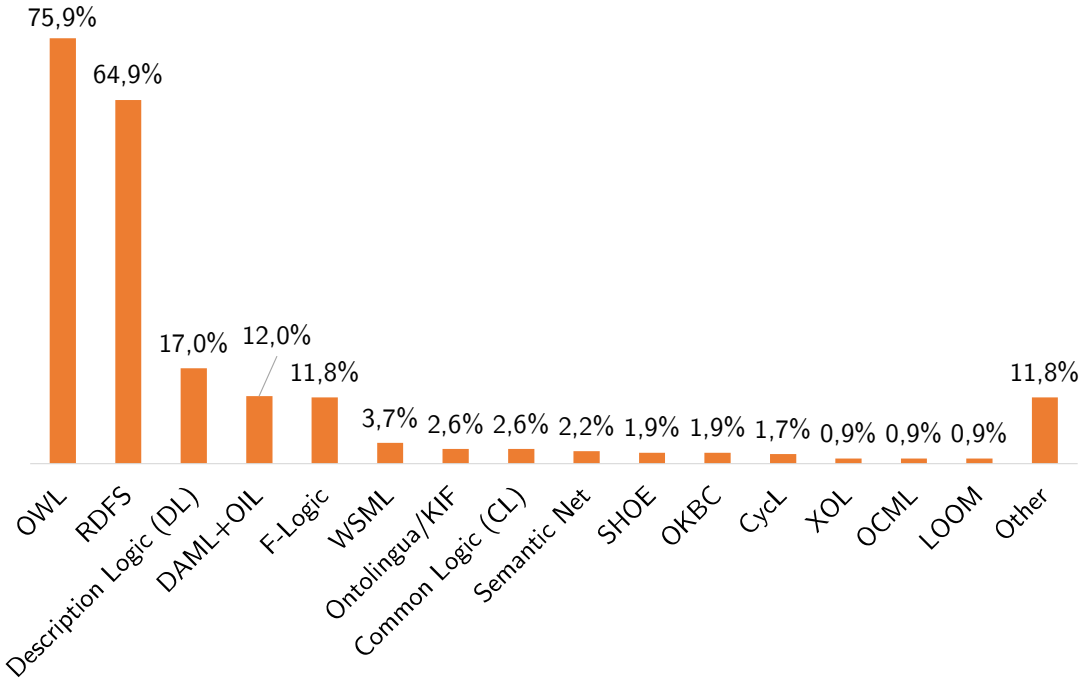


Figure 17: Usage of ontology languages 2006/2007 (cf. Cardoso, 2007)

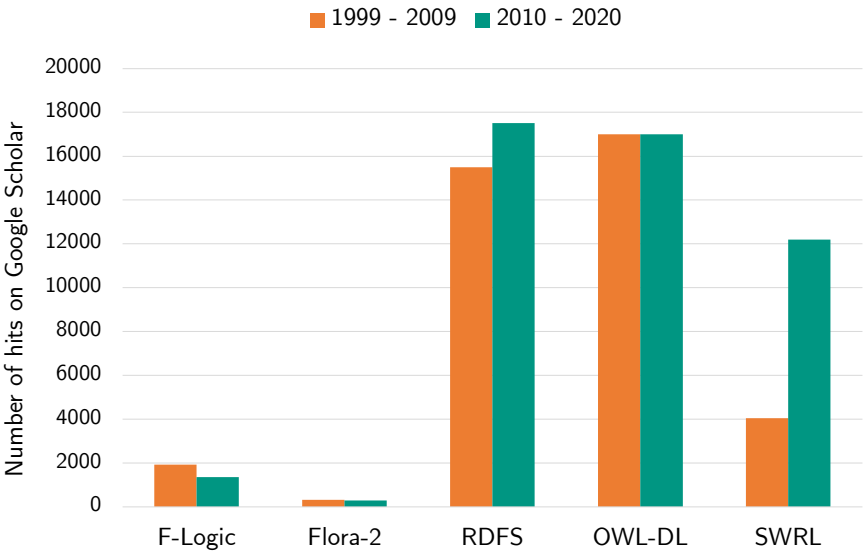


Figure 18: Number of hits on Google Scholar (query on 28.11.2020)

Table 6: Comparison of the RDFS, OWL 1, and OWL 2 (cf. Munir and Sheraz Anjum, 2018)

Concepts	RDFS	OWL 1	OWL 2
Formal semantics	✓	✓	✓
Equivalence	-	✓	✓
Class definitions	-	✓	✓
Constraints	-	✓	✓
Enumerations	-	✓	✓
Cardinality constraints	-	✓	✓
Inference	-	✓	✓
Property chains	-	-	✓
Disjoint properties	-	-	✓
Qualified cardinality restrictions	-	-	✓

3.9 Classification of reasoning techniques

One of the main tasks of a knowledge-based system is to draw conclusions from existing knowledge representations (cf. Styczynski et al., 2017). Brachman and Levesque (2004) state that the task of reasoning is the computation of the entailments (logical consequences) of a knowledge representation. An *inference* is a conclusion drawn from the inference process conducted by an inference mechanism. Reasoning methods can be classified as *inductive*, *deductive,* or *abductive* (cf. Folger and Stein, 2017). Figure 19 shows a classification of different reasoning techniques. Knowledge-based and data-driven reasoning can be distinguished. While knowledge-based reasoning can be divided into the subcategories *deductive* and *analogical* reasoning, data-driven reasoning is usually inductive. Further method-specific details are described in the following.

Knowledge-based reasoning paradigms use deductive reasoning approaches, which start from a general rule or fact to derive specific statements (cf. Styczynski et al., 2017). Causal rules and relationships form the basis for deductive reasoning. An inference engine implements a reasoning model representing these rules or relationships to use a knowledge representation (cf. Latombe, 1987). Furthermore, analogical reasoning methods use analogies to conclude (cf. Styczynski et al., 2017).

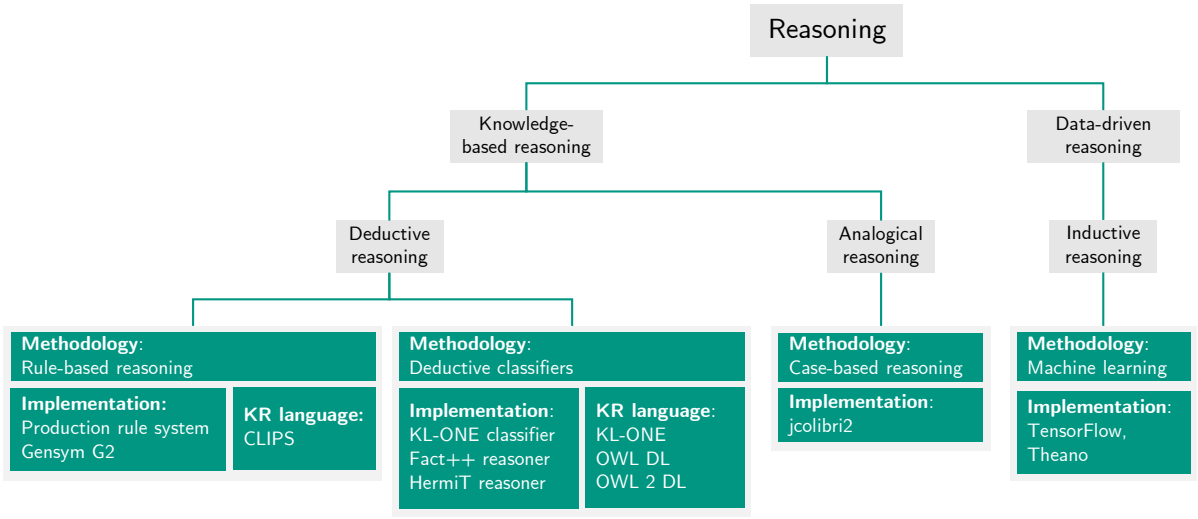


Figure 19: Overview of reasoning techniques

Analogical reasoning is based on heuristics (e.g., cases). The type of knowledge representation impacts the reasoning or inference mechanism and vice versa. Data-driven reasoning, on the other hand, is usually inductive. Hence, general rules are derived from observed individual circumstances, such as measured values or other data (cf. Styczynski et al., 2017). The most relevant reasoning techniques in the context of this work are described hereafter.

3.9.1 Rule-based reasoning

Rule-based reasoning (cf. production rules, Figure 12; Figure 19) is a widely used reasoning paradigm. The reasoning architecture is based on a *rule base* and an *inference engine* (cf. Dutta and Bonissone, 1993). The rule base consists of a domain-specific set of IF-THEN rules (cf. Section 3.6). The IF part of the rule is called a *premise,* while the THEN part is called a *conclusion.* The inference engine deduces information from the knowledge base via an inference mechanism, such as forward and backward chaining. Forward chaining is a bottom-up method. It starts with conditions or facts to deduce a conclusion. According to Al-Ajlan (2015), it "starts with a set of known facts and applies rules to generate new facts whose premises match the known facts." Backward chaining is a top-down method, which means it starts from a conclusion or hypothesis and searches for rules (conditions) that lead to that conclusion (cf. Al-Ajlan, 2015). Since 1986 rule-based reasoning (RBR) has been implemented by various researchers using the expert system shell Gensym G2 (cf. Kang et al., 1999). Other researchers used the rule-based programming language CLIPS to implement a reasoning engine (cf. Wygant, 1989).

3.9.2 Deductive classifiers

Deductive classifiers are concerned with dedicating logical consequences that arise from declarations in the form of related concepts. Deductive classifiers are also called semantic reasoners or simply reasoners. Brachman and Schmolze (1985) state that a (deductive) classifier determines the subsumed relationships between a newly specified concept and other concepts in a network.

Drummond *et al.* (2006) describe subsumption as a superclass-subclass relationship, which means "all members of a subclass are members of its superclasses." An example subsumption is shown in Figure 20, where 'Risk management' is a superclass of HAZOP; hence, risk management subsumes 'HAZOP'. According to Schmolze and Lipkis (1983), a classifier "compares two concepts and decides whether or not the first subsumes the second." Hence, a classifier places concepts properly and establishes hierarchical links between them (cf. Schmolze and Lipkis, 1983).

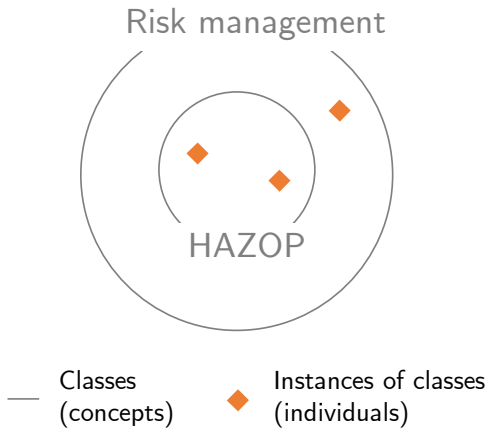


Figure 20: Subsumption explained

One of the first *deductive classifiers* was the KL-ONE classifier (cf. Brachman and Schmolze, 1985), based on the frame-based (cf. Section 3.6) KL-ONE knowledge representation system.

Within the context of ontologies, classifiers are usually simply called *reasoners*. Glimm *et al.* (2010) state that a core reasoning service provided by reasoners is the computation of subsumption hierarchies of classes and their relationships called *ontology classification*. Ontology classification is required to derive conclusions from asserted logical facts and axioms of an ontology. Wang *et al.* (2006) describe this coherence as "the classifier tries to build a model that satisfies all the axioms in the ontology."

Established reasoners for evaluating ontologies are *FaCT++*, *HermiT*, *Pellet*, and *Riverport* (cf. Tsarkov and Horrocks, 2006; Tsarkov and Horrocks, 2006; Sirin *et al.*, 2007; Haarslev *et al.*, 2012). Furthermore, reasoners can identify logical inconsistencies in the axioms, reduce redundant information, and find knowledge conflicts (cf. Abburu, 2012). Schmolze and Lipkis (1983) describe that automatic classification provides the means to enforce network semantics and check the consistency of the description.

3.9.3 Case-based reasoning

Aamodt and Nygård (1995) describe case-based reasoning (CBR) as an artificial intelligence (AI) paradigm for problem-solving. It is a pattern-based problem-solving method. The central idea behind CBR is the drawing of conclusions by analogy. Therefore, knowledge about past situations (cases) is reused in new situations (cases). These cases are composed of case attributes used to calculate a similarity measure between attributes and entire cases. Similarity measures are usually numerical values within the range [0...1]. A similarity measure of 0 means there is no similarity, while 1 means the compared cases are equal. Various similarity measurement methods can be used to compare different data types (e.g., objects, strings, or sets) (cf. Zhao et al., 2009). Thus, the calculation of the similarity measure depends on the data types of the attributes (e.g., string, int). Every time a case is successfully solved, the solution is added to a collection of case data, and this is how these systems learn. The use of CBR in the scope of this dissertation is presented in detail in Section 5.13.

3.9.4 Data-driven reasoning

Besides knowledge-based reasoning, data-driven reasoning techniques represent inductive reasoning techniques (cf. Figure 19). A data set is classified using a mathematical representation (model) during reasoning. Predictions can be made based on that model based on the input data. Data-driven reasoning techniques typically include *machine learning* methods to identify patterns or generate activity models using training data (cf. Sukor et al., 2019).

Usually, three types of machine learning are distinguished: *supervised learning*, *unsupervised learning,* and *reinforcement learning* (cf. Buxmann and Schmidt, 2019). A prediction model is generated in supervised learning based on a training data set and evaluated using a test data set. Typically, classification or regression models are generated using supervised learning methods. In contrast, unsupervised learning methods do not require a training data set. This means that the algorithm has to categorize itself (cf. Buxmann and Schmidt, 2019, pp. 10). Unsupervised learning methods are predestined to identify patterns, e.g., clustering. The idea of reinforcement learning

is to learn an optimal strategy for a defined problem (cf. Buxmann and Schmidt, 2019, pp. 11). During reinforced learning, the algorithm receives feedback in a reward or punishment. The choice of methods depends on the data sets to be analyzed and the task to be performed, e.g., classification or regression. Due to many available paradigms and methods, a suitable approach must always be chosen in dependence of the task at hand.

3.9.5 Advantages and disadvantages of reasoning techniques

The type of knowledge representation (KR) directly affects the reasoning or inference mechanism and vice versa. Accordingly, after selecting a KR scheme, a group or a specific reasoning technique results. Table 7 shows an overview of the advantages and disadvantages of selected reasoning techniques. The advantage of rule-based reasoning (RBR) is the low computational cost, and the derived conclusions are comprehensible. Rule-based approaches are widely used, and even parts of the SWRL language (cf. Section 3.8.1) are based on rules. One of the major drawbacks is that the problem statement must be matched precisely by the rules. Case-based reasoning (CBR) is beneficial in the case of new unknown cases (new cases that are not explicitly modeled) and in case of incomplete information (not all case attributes are defined). Prentzas and Hatzilygeroudis (2007) refer to such as "handling unexpected or missing input." Also, the reasoning abilities of CBR increase with every added case. The determining characteristic of CBR is the similarity measure, which depends on the input object (case) and its attributes. The determination of the similarity measure depends on the involved data types. Additional advantages and disadvantages of CBR are shown in Table 7.

Within data-driven reasoning, conclusions are derived based on raw or preprocessed data. There is no need for a priori knowledge, such as rules or cases, to apply data-driven reasoning methods. The inference is based on collected or measured data. Typically, the data collection and preprocessing involve considerable effort since every data set must be labeled for drawing conclusions. The model can be easily (re)trained based on new data if data changes. Not every problem can be addressed with data-driven reasoning. Typically, it is used for classification, pattern recognition, or behavioral prediction.

The advantages of deductive classifiers are that the semantic context and implicit knowledge embedded within the KR are considered. However, applying deductive classifiers requires a complete domain model where incomplete information leads to wrong conclusions. In addition, detailed reasoning can be slow. As described in Section 3.8.4, OWL (DL) related languages are particularly important in this research work. Hence, reasoners that can manage these languages are particularly relevant, such as DL reasoners. Logical consistency is essential in complex ontologies because inconsistencies make ontologies unusable. Another reasoning service of a DL reasoner is classification. Classification means that subsumption relationships between classes and properties are computed. Thus, unexpected subsumptions can be automatically identified (cf. Wang et al., 2006). The most widespread reasoners for OWL-based ontology languages are:

- **OWL**: HermiT, Pellet, Fact++, Racer, KAON2,
- **SWRL**: Pellet, Racer, KAON2,
- **OWL 2**: HermiT, Pellet, ELK.

Gonçalves et al. (2013) evaluated reasoners such as Pellet, HermiT, FaCT++, and JFact (Java porting of FACT++) and quantified the robustness and performance of reasoners. Gonçalves et al. (2013) processed 1071 ontologies and found out that the Pellet reasoner is the most robust. Furthermore, they state, "HermiT and FaCT++ have the same overall robustness, but FaCT++ has fewer errors and higher impatient robustness." Additionally, they note that statements about robustness and performance strongly depend on the type of ontology and cannot merely be generalized. Thus, the reasoning performance depends on the ontology structure, including the axioms used, ontology size, and modeling style, e.g., the use of classes and individuals.

Table 7: Advantages and disadvantages of different reasoning techniques

Reasoning	Advantage	Disadvantage
Rule-based	Low computational costs	Rules must match the problem statement exactly
	A sophisticated rule engine can be used	Reasoning fails in case rules do not unambiguously cover the domain
	The conclusions drawn are comprehensible	The meaning and semantic context of rules is not always represented
	-	For large rule bases, rule-based inference can lead to scalability issues; (cf. Prentzas and Hatzilygeroudis, 2007)
Case-based	No complete domain model needed	A large number of cases (case base) is required for useable results.
	Quality of conclusions improves with an increasing number of cases	The determination of an appropriate similarity measure requires expert knowledge within the relevant domain
	Low maintenance due to new cases	Similar cases can lead to ambiguous conclusions
Data-driven	The inferences can be drawn without any specifications in knowledge/rule/case bases	Typically, data set must be preprocessed, e.g., labeled, and the feature space must be defined
	Inference is based on real/measured data, not on a surrogate model within a domain	Predictions/inferences depend on the quality of the data
	A model can be easily (re)trained with new data	Not dependable outside the data range
	-	Transferring results/findings to other processes or domains is challenging/not possible
Deductive classifiers (ontology reasoners)	Semantic context is considered	"No immediate explanation of the ability to reason about the unknown" is offered; (cf. Johnson-Laird, 1999)
	Implicit knowledge can be inferred	A complete domain model is required
	-	Detailed reasoning can be slow

3.10 Summary of the fundamentals

This chapter presents hazard identification methods commonly used in the chemical process industry. The HAZOP method and its advantages and disadvantages are described in detail. The description includes method-specific aspects such as the causation model on which the HAZOP method is based, factors influencing the obtained results, documentation of results, and risk assessment of scenarios. In order to be able to analyze and evaluate existing approaches and prototypes from the literature, basic terms such as data, information, and knowledge are distinguished. Information technology fundamentals such as graph theory, knowledge representation formalisms, automatic reasoning are presented. Automatic reasoning paradigms serve to make formal knowledge of knowledge representation usable. Thus, the reasoning paradigm depends directly on the knowledge representation formalism. Since the representation and use of knowledge have a significant role in this work, the advantages and disadvantages of knowledge representation formalisms and reasoning paradigms are presented. The focus is particularly on knowledge-representation formalism ontologies, ontology languages, and reasoning. Due to their expressiveness and formal semantics, ontologies are particularly suited to represent required expert knowledge. The ontology language OWL 2 DL proves to be particularly suitable to represent complex interdisciplinary knowledge due to its expressiveness, widespread usage, and available reasoners (cf. 3.8.4). This chapter covers the objectives from Chapter 2 (1).

4. Evaluation of the state of research

Numerous approaches have been developed, and prototypes have been built for automating aspects of the HAZOP analysis. Accordingly, the state of research must be comprehensibly reviewed within the scope of this dissertation. Therefore, this chapter analyzes and evaluates all past and contemporary approaches of different research groups. Subsequently, a summary of the results is presented, and the resulting research gaps for this research work are derived based on the findings.

4.1 Approaches from the literature

The following chapter is mainly based on the findings of Single et al. (2019b) and expanded by more recent findings. Various approaches are evaluated in this work according to their method-specific aspects, reasoning techniques, modeling approaches, knowledge representation capabilities, and usage of external software.

4.1.1 HAZOPEX and HAZOPTool

Suokas et al. (1990) developed an expert system to support safety analysis called HAZOPEX. The expert system seeks to identify potential causes of process deviations. It was implemented using the KEE expert system shell (cf. Heino, 1998). Based on HAZOPEX, Heino et al. (1992) developed HAZOPTOOL. The approach includes system description, knowledge management, reasoning system, analysis documentation, and reporting (cf. Heino et al., 1995; Heino, 1998). It is based on an object-oriented plant description and general knowledge about the process units. Various knowledge bases exist, such as chemical substance knowledge base, chemical reaction knowledge base, and unwanted chemical reaction knowledge base.

The model concept of a process unit is shown in Appendix Figure U-1. A process unit has input and output deviations, external and internal events, and consequences. Also, multiple chains of events are allowed. The output deviations and consequences can be caused by input deviations, external and internal events. Process information, such as process structure, reactions, and chemicals, is used to deduce the causes and consequences of process deviations.

The reasoning is rule-based and is conducted unit by unit. For each unit type, a set of rules was defined. The rules assigned to the units are used by the reasoning mechanism that consists of three reasoning methods: backward chaining for causes, forward chaining for consequences, and propagation procedures. There is a user interface to modify and extend the information in the reasoning module. Most of the implemented rules within HAZOPEX are related to causes (rule: 330 causes, 20 consequences, 20 measures). Also, the reasoning was not conducted based on the fluid properties within HAZOPEX. Heino (1998) mentions that there was no actual evaluation of the prototype within the HAZOPTOOL project. Thus, no benchmark results are publicly available. HAZOPTOOL did not consider safeguards, which are a crucial aspect of the HAZOP method. The focus of the HAZOPEX expert system was to suggest potential causes based on deviations. Accordingly, this approach covers only partial aspects of the HAZOP method.

4.1.2 Qualitative Hazard Identifier

Catino and Ungar (1995) developed a prototype hazard identification system called qualitative hazard identifier (QHI). Qualitative differential equations (QDE) are derived from differential and algebraic equations for predicting process and plant behavior. QPC is a model builder that transforms physical models into QDEs for qualitative simulations (Crawford *et al.*, 1990). QSIM is a tool to conduct qualitative simulations of QDE-based models. It was developed by Kuipers (1987) and was described by Janowski (1987). The QHI approach requires a library of QPC models that

describe the process, faults, and hazards. It also requires a symbolic representation of the process units, their connections, and the operating conditions.

A schematic description of the QHI approach is shown in Appendix Figure U-2. First, QPC is used to build QDE models based on a library containing physical and chemical phenomena and a physical description of the plant. Afterward, QHI uses a pre-processor to enhance the model by disturbance functions or rules with the scope of perturbing hazard-relevant parameters. Thus, a fault library is used that contains generic faults, such as leaks, blocked pipes, or controller failures. Afterward, QSIM runs the simulation while QHI analyzes the system to detect dangerous behavior. The physical description (process plant and process description) is compared with a fault from the library of faults during analysis. If there is any match, the simulation model is perturbed to represent the fault. Afterward, the simulation is run while the behavior of the system is evaluated. A potential hazard is identified if the perturbation in the simulation model leads to an event (e.g., overpressure or overtemperature). A case study involved a reactor section of a nitric acid plant. The quality and completeness of the identified hazardous events depend directly on the used libraries, such as the fault library.

The authors of QHI state that the quantitative simulations scale better than QDE simulations if the plant equipment is densely connected. The extension of the QDE models by disturbance functions and rules requires a deep understanding of the models and qualitative physics (cf. Kuipers, 1987). Thus, the effort of identifying safety-relevant scenarios using QDE based simulations is high, depends heavily on the quality of the underlying model, and cannot be directly transferred to other processes and plants. This is shown because the authors focused on specific systems, e.g., nitric acid plant. The application of this methodology is limited to well-known chemical processes. It is difficult to base HAZOP studies on qualitative simulations, given the large number of scenarios possible. Abnormal scenarios that are not based on process parameters and the model structure cannot be identified, e.g., operational regulation, external disturbance, or mechanical damage. Accordingly, qualitative process simulation-based approaches cannot solely be used to identify plausible scenarios. Furthermore, the method tries to identify faults and associated hazards. The identified scenarios must be transferred manually into HAZOP worksheets, including their categories. Furthermore, no safeguards are identified in the approach.

4.1.3 ExpHAZOP+

The research group around Khan et al. developed various approaches: OptHAZOP by (Khan and Abbasi, 1997b), TOPHAZOP by (Khan and Abbasi, 1997a), EXPERTOP by (Khan and Abbasi, 2000), and ExpHAZOP+ by (Rahman *et al.*, 2009). The ExpHAZOP+ approach consists of a graphical user interface, a knowledge base, and an inference engine. A graphical user interface is used to draw a process flow diagram.

Relevant knowledge is divided into general and process-specific knowledge. General process knowledge includes knowledge, e.g., hazardous characteristics of substances, physical and chemical properties, or physical states of chemicals. The process-specific knowledge is based on frames and rule networks and includes operation, equipment, and substances data. Based on user-selected deviations, the inference engine searches for causes and consequences using a fault propagation algorithm to propagate deviations through the system. The user needs to specify the starting and ending equipment. In the case of multiple paths, the user needs to specify a specific path to be investigated. The fault propagation algorithm uses forward and backward search techniques to find equipment connections to identify causes and consequences.

Rahman et al. (2009) assume that the causes of equipment failures do not change significantly from facility to facility. No information is given about the level of detail to which equipment is represented. Slight differences in equipment such as other fixtures or instrumentation can lead to other causes, although the equipment is generally similar. It is also assumed that consequences are functions of process type, process condition, and involved substances. Rules are used for

reasoning purposes, which can cause ambiguous results in case rules cannot be matched precisely to the problem statement, e.g., an atypical configuration of process equipment. Rule-based knowledge representations can be problematic if complex facts or statements are represented that do not precisely match a rule (cf. Table 4). This issue applies particularly to the general process knowledge of the knowledge base of ExpHAZOP+. Furthermore, the method does not consider safeguards, which is an integral concept of the HAZOP method. Also, the intended function of equipment and nodes is not considered explicitly.

4.1.4 PHASuite

The research group around Venkatasubramanian developed *HAZOPdigraph* models. Based on it, the following approaches were developed: Petri Net-Digraph, HAZOPExpert, Batch HAZOPExpert, and PHASuite (cf. Vaidhyanathan and Venkatasubramanian, 1995; Srinivasan and Venkatasubramanian, 1996; Vaidhyanathan *et al.*, 1996; Srinivasan and Venkatasubramanian, 1998; Zhao *et al.*, 2005b). The most recent approach PHASuite is described in detail below.

Zhao et al. (2005b) distinguish between declarative and control knowledge within the knowledge representation. The declarative knowledge is stored in semantic networks, frames, predicate logic, and production rules and saved in Microsoft Access databases. Additionally, the control knowledge, which represents procedural knowledge, is hardcoded in C++. Conclusions are inferred using a two-layered model-based reasoning technique. The safety model layer (the first layer, cf. Figure 19) contains hazards, and the colored *Petri net layer* represents connection information. The safety model layer describes causal relationships between process variables using digraphs. There are connectivity ports between models and rule sets concerning local causes and consequences of deviations. Safety models are stored as Microsoft Access databases.

Colored Petri nets (cf. Section 3.4) are used in the second layer due to their strong expressiveness (cf. Figure 21). They consist of *places*, *transitions, edges,* and *colored tokens* carrying data (cf. Figure 21). A transition is enabled if the input contains at least one token. These tokens are propagated through the system. An input token is consumed when a transition is activated, while an output token is produced. There are three types of tokens: *material propagation*, *consequence*, and *cause*. Using Petri nets, discrete event systems, e.g., batch processes, can be modeled since changing process states can be directly modeled. The system analyzes and calculates the operation and equipment levels with different scopes.

During the reasoning process, the local effects of deviations on all process variables are investigated, and the outlet condition of the equipment is determined. Afterward, it is propagated from equipment to equipment until the end of a current path. Therefore, local deviations are propagated first on the operation level and afterward on the equipment level. During consequence analysis, tokens are propagated downstream while propagated upstream during cause analysis (cf. Zhao et al., 2005b). The Petri net layer propagates material and deviations, and the safety model layer generates HAZOP-related results.

Additionally, case-based reasoning is used to reuse solutions from past cases. The case base comprises generic models divided into operation and equipment models on different layers. These models share similar properties, feature types, subtypes, and structural differences, e.g., components. Models are indexed based on case attributes to derive a similarity measure to compare different cases. Based on this indexing, the best matching case is identified. There are software-based interfaces for the information exchange between PHASuite and commercial software packages. PHASuite shares process information with Batch Plus (AspenTech, n.d.), e.g., equipment, material, and reaction information. Ontologies are used to eliminate conceptual and terminological confusion, exchange information, and match vocabulary. For instance, the term “Rpm” from BatchPlus is translated into “AgitatorSpeed” in PHASuite using an ontology.

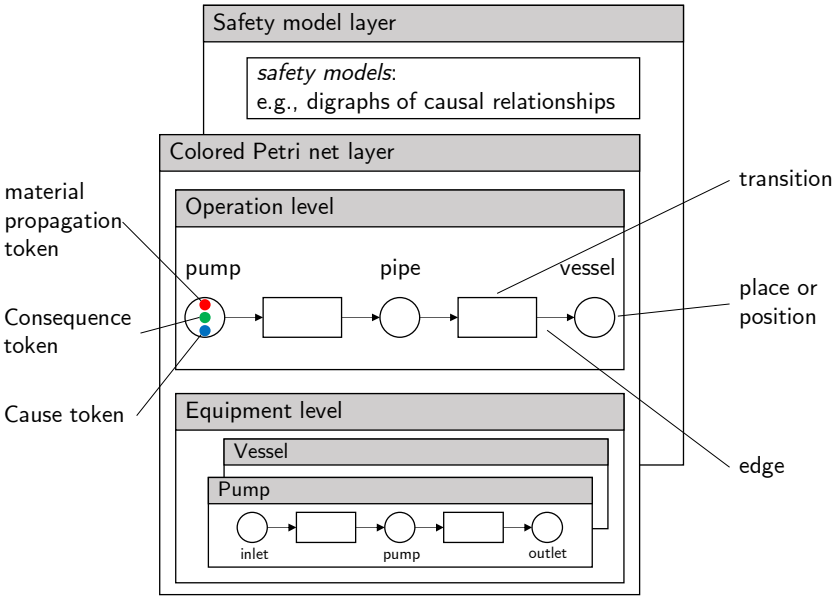


Figure 21: Schematic two-layered reasoning engine (cf. Single et al., 2019b)

PHASuite has its own P&ID drawing facilities for equipment information and process flow diagrams (PFD). Using colored Petri nets combined with a two-layer, two-level model-based reasoning methodology combined with case-based reasoning delivers strong reasoning capabilities. As a result, different conclusions with different objectives can be drawn based on different representations. The idea of exchanging information between process simulators, computer-aided HAZOP systems, and HAZOP documentation software is future-oriented. On the other hand, frames and semantic networks lack formal semantics that limits knowledge representation capability (cf. Table 4). This situation could be improved by using ontologies for improved semantics. Ontologies have just been used for matching vocabulary, but no further information regarding the structure was provided. Furthermore, safeguards are only considered as a supplementary aspect.

4.1.5 AutoHAZID

One of the most conceptually mature and extensively documented prototypes from the past is AUTOmatic HAZard IDentification (AutoHAZID) developed within the STOPHAZ project (cf. McCoy et al., 1999). Its purpose is to identify problems before a HAZOP meeting (cf. Wakeman et al., 1997). It is based on the insights of the HAZOPTOOL project (cf. Section 4.1.1). The process plant is represented using connected unit models and a fluid model system. The unit models are based on frames (cf. Section 3.6), while the connectivity of the units is represented via SDGs (cf. Section 3.4). The fluid model contains information regarding the behavior of fluids and fluid properties.

A HAZOP emulation algorithm was designed to identify scenarios using an inference engine. A backward search identifies causes of deviations using 'fault paths'. If causes have been found, consequences associated with the deviation are identified. The fluid model system is rule-based and verifies or rejects causes and consequences based on fluid properties. AutoHAZID incorporates qualitative propagation relationships in functional equation models to represent the propagation of deviations, faults, and consequences. These functional equations are used together with initial event statements (e.g., blockage, leak) to describe how propagations were initiated. The approach also includes a consequence evaluation system for quantitative consequence analysis to quantify the severity and estimate the rank of potential loss events. Furthermore, a rule-based module checks the plant configuration for errors, e.g., a vessel not connected to a safety valve. The results

are presented in a HAZOP worksheet with the columns: deviations, causes, consequences, and protections. Multiple filters are applied to reduce output volume and suppress repeating scenarios.

The description and explanation of the concepts, implementation and capabilities of HAZID are exemplary, and findings are used in other approaches. The inference mechanism is based on rules. Rule-based KR can be problematic if complex facts or statements are represented that do not precisely match a rule (cf. Grimm et al., 2007). Multiple filters are applied to reduce the output volume, indicating that the inference mechanism can be refined. The underlying causal model assumes that causes and consequences are linked, which must not always be the case. According to the authors' statement, "HAZID did not perform very well in finding protections" (cf. McCoy et al., 2000).

4.1.6 PetroHAZOP (HAZOPSuite)

Zhao *et al.* (2009) proposed a HAZOP expert system, where case-based reasoning and ontologies for knowledge representation are used. According to the authors, it is designed to improve the limited capabilities of PHASuite regarding *non-routine* HAZOP analysis (cf. Section 4.1.4). Six ontologies are used for knowledge representation:

- Process ontology, e.g., classification of chemical plant types,
- Process unit ontology, e.g., descriptions of process units (OntoCAPE),
- Unit operation ontology, e.g., enthalpy change or separation (OntoCAPE),
- Equipment ontology, e.g., equipment is specified based on process parameter and structural specifications,
- Material ontology, e.g., chemical species information, material safety sheet data,
- HAZOP ontology, e.g., guidewords, deviations, causes, consequences, safeguards.

The process and operation ontologies are based on OntoCAPE by Morbach *et al.* (2009), while ideas from Batres *et al.* (2005) are considered during the design of the ontologies. A CBR algorithm determines a similarity measure between cases during the reasoning process to find the closest matching cases. There are various methods to determine the similarity measure, depending on the cases' data types (cf. Section 3.9.3).

A hierarchical case structure is introduced to facilitate the case-based reasoning technique. Each case is characterized by equipment, design parameters, substances, operating conditions, and stream context conditions. A knowledge maintenance module is used to maintain and extend cases. The process plant and its equipment are modeled via a graphical user interface where nodes, equipment, process variables, and deviations are entered in lists and tables. The system is implemented using Java, and all cases are stored in relational databases. According to Zhao *et al.* (2009), the systems contain around 900 cases. Cui *et al.* (2012) state that the software is deployed and used within China national petroleum corporation (CNPC) for more than ninety refinery and petrochemical processes. In the course of advancement, the approach HASILT combines HAZOP with the layer of protection (LOPA) analysis, safety requirements specification (SRS), and safety integrity level (SIL) (cf. Cui et al., 2012).

The usage of ontologies for knowledge representation entails state-of-the-art knowledge storing capabilities. Zhao *et al.* (2009) state that they created several ontologies within their approach. Nevertheless, they make no further statements on the structure, components, or ontological model. One of the few descriptions of the ontological model is that the ontological concepts are hierarchically linked via inheritance relationships. On the other hand, non-inheritance relationships play a significant role in expressive ontologies.

Using this methodology, the correctness of the conclusions drawn depends strongly on the capabilities of the similarity matching algorithm since it determines the similarity measure between cases. One category of the case base is the equipment, including its design parameters. Further information regarding the equipment is needed to precisely determine the degree of

similarity, e.g., instrumentation, fixtures, apparatus. Otherwise, this leads to a substantial generalization of the cases, which applies to the PetroHAZOP approach. Since the cases are distinguished based on *equipment*, *substances*, *operating conditions,* and *stream context conditions*, it is assumed that causes, consequences, and safeguards of individual cases are always related. Accordingly, the possibility of determining causes independent of consequences and vice versa is missing, leading to a static view on scenarios.

The characterization of the scenarios, e.g., cause of a deviation within the case base, directly impacts the similarity measurement between cases. Also, a definition is needed, which degree of similarity is accepted to consider the case credible (cf. threshold, Section 5.13.1). Therefore, it is questionable if the methodology can be directly transferred from oil and gas to chemical processes with an immense diversity of plants. The intended function of nodes is not considered to a sufficient extent. Furthermore, the propagation of deviations or events is not considered within this approach.

4.1.7 LDGHAZOP

Cui *et al.* (2008) introduced the Layered Digraph (LDG) methodology. It aims to improve the representation of relationships between process variables compared to an SDG-based representation. Cause and effect relationships between process deviations are qualitatively modeled considering three dimensions: process variables, process deviations, and interactions of deviations (cf. Cui et al., 2010). An exemplary LDG model is shown in Figure 22. In every layer, all process variables, such as, Q_{in} are modeled using digraph models. For every additional guideword, another layer is added, e.g., *more*, *less*, *none*. There are interconnected nodes between these layers. In addition to *nodes* and *edges*, there are three types of *tags*: cause, consequence, and comment. They are attached to nodes and contain information regarding abnormal causes or adverse consequences. Furthermore, there are three types of tags: cause, consequence, and comment.

LDG models are linked via deviation image nodes (cf. Figure 22). These are used to model a process plant that consists of various LDG models. New LDG models can inherit knowledge from existing models stored in a model library. The LDG model can be represented by a three-dimensional relational matrix stored in a table (cf. Cui et al., 2008). Also, causes, consequences, and interconnections among equipment are stored using an SQL-based database management system. The reasoning engine LDGExpert propagates deviations and deduces consequences in the equipment network. It uses the cause and consequence tags to support the reasoning mechanism. Cui *et al.* (2010) integrated Intergraph Smart Plant P&ID with their LDGHAZOP approach. Therefore, the P&ID data is extracted using the Intergraph SPPID Application Programming Interface (API) called *Llama*. Afterward, suitable LDG models from the LDG model library are matched to the equipment found in P&IDs. The P&ID interconnections are used to link the selected LDG models.

The advantage of the method is that the interactions between various deviations are captured qualitatively. Cui et al. (2008) state that the number of automatic reasoning results within LDGHAZOP is much more than experts' results, but most are meaningless. The results generated by LDGHAZOP are still far from completion before a HAZOP practitioner validates and modifies them. This incompleteness stems from multiple LDG models needing to be coupled for representing chemical process plants. LDG models contain many process variables (one additional dimension for every process variable) propagated through the plant sections leading to redundant or ambiguous causes and consequences. Besides propagation, rules manage cause and consequence relationships via tags. Tags explain the contexts, serve as constraints, and are attached to the nodes of the LDG models. Causes and consequences are not (semantically) related, making it difficult to generalize. Furthermore, causes and consequences must be re-modeled for each LDG model and cannot be easily transferred between models.

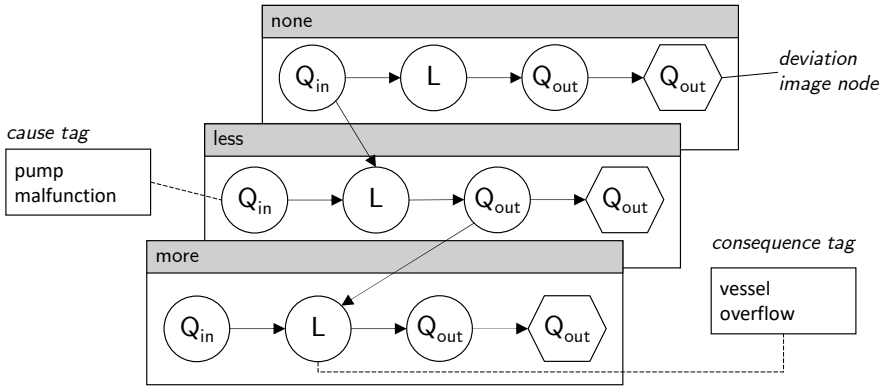


Figure 22: LDG model of three guidewords and three variables (cf. Single et al., 2019b)

This effort must be made because causes and consequences are indirectly linked to the equipment via the model, and there is no explicit semantic relationship between them. Also, there are no general links to the properties of involved substances. The method-specific modeling effort could be reduced by using ontologies to relate concepts to each other or inherit concepts. According to Yang *et al.* (2010), the implementation of SDG representations takes time, and they are hard to maintain. LDG models are even more complicated compared to SDG models. Finally, safeguards, risk assessment, and the intended function of nodes are not considered, which are integral parts of the HAZOP method.

4.1.8 MFM HAZOP Assistant

The MFM HAZOP assistant is based on multilevel flow modeling (MFM), which was introduced by Lind (1981) and adapted by Rossing et al. (2010). It is a framework based on functional modeling, describing the purpose and function of process systems and their causal relationships. MFM is based on hierarchical abstraction (cf. Rasmussen, 1985). The MFM HAZOP assistant is intended to support traditional HAZOP studies. Knowledge regarding plant functions is translated into an MFM model considering different abstraction levels. A simple MFM model consisting of source, vessel, safety valve, and sink is shown in Figure 23. Further details regarding the modeling syntax can be found in Lind (2004).

If a process equipment model is not included in the model library, it can be modeled using the MFM model builder. Hence, the plant needs to be divided into sections and nodes. Afterward, the process variables and deviations must be specified. The reasoning method is based on the rule engine JESS and the MFM model. The reasoning system generates causal paths and root causes for faulty scenarios. It identifies causes via reasoning paths of deviations. Rules automatically generate these reasoning paths in the MFM reasoning engine (cf. Lind, 2011). Rossing et al. (2010) propose multiple rule-bases:

- causal rules: to generate fault and consequence trees,
- goal reasoning: to generate goal trees and
- rules: regarding counter and preventive measures.

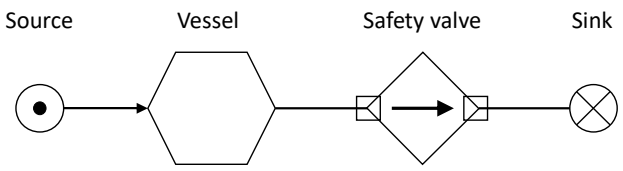


Figure 23: Exemplary MFM model (cf. Single et al., 2019b)

Currently, Microsoft Visio is intended to apply the MFM modeling language to model process plants. The focus of the approach is on the identification of root causes of scenarios. At this stage, the automation of the HAZOP is not in the scope of the method. The authors state that the reasoning engine can be extended to identify possible consequences (cf. Rossing et al., 2010). The results of the approach are causal trees of suggested causes. Accordingly, no complete scenarios are identified. The content of the causal trees must be transferred into the HAZOP worksheet. Overall, it is an expressive modeling language to model process plants, including their processes and operations. Besides, the aspects safeguards and risk assessment are not addressed. The prototype cannot directly be used to automate the HAZOP method in its present form.

4.1.9 D-higraph HAZOP assistant

Rodríguez and De la Mata (2012) introduced the qualitative modeling technique D-higraphs (dual higraphs) to support HAZOP studies. Here, process plants are described considering structural (e.g., process variables), behavioral (e.g., system behavior), and functional (e.g., connections between components) aspects. Since it is based on graphs, there are nodes and edges, but nodes are called blobs, while an additional formalism is introduced. These blobs consist of individual elements: *function*, *actor,* and *condition*. An exemplary D-higraph representation of an exemplary reactor and safety valve system is shown in Figure 24. The function element describes the blob function, while the *ACTOR* represents the equipment or system. The *condition* represents dependencies; for instance, the level L is influenced by the feed flow Q_1 and the flow to be discharged Q_2. Additionally, there are different types of edges: *mass* (solid line), *energy* (dashed line), and *information* (dotted line). Another level of the hierarchy is added using different colors. Process blobs are green, and control blobs are colored orange. D-higraphs represent specific knowledge about the plant, while the hierarchy and relationships are explicitly modeled.

The reasoning engine is implemented using the rule-based programming language CLIPS (cf. Wygant, 1989). Hence, the system uses a rule base for causal reasoning to infer causal and consequence trees for process deviations to detect potential hazardous events. Before a system can be analyzed, all process units, process variables, and relationships need to be modeled in a drawing tool using the D-higraph formalism. According to the authors, they work on integrating the method with Aspen Plus. Mechhoud et al. (2016) applied the D-higraph HAZOP Assistant to a process plant, including a high-density polyethylene reactor. Furthermore, they compared the D-higraph method to the structured analysis and design technique (SADT). It is a functional analysis method that can break down functions hierarchically. The D-higraph HAZOP assistant integrates functional and structural information within a single model.

The D-higraph formalism is an expressive methodology to model the process plant and is simultaneously well-suited to model the relationships between process variables. The approach is similar to the MFM HAZOP assistant, where a different qualitative modeling language is used (cf. Section 4.1.8). Due to the complexity of the modeling language, there is a chance that the modeling of large chemical process plants is time-consuming. The D-higraph HAZOP assistant provides potential causal trees of deviations. The approach is conceptual, and the results presented involve deviations and causes, but not entire scenarios. The results must be interpreted and do not provide a complete description of the scenarios. Safeguards and risk assessment are not considered within this approach.

4.1.10 Other Approaches

Rootsaert and Harrington (1992) developed a rule-based prototype called COMHAZOP to identify hazards. The approach consists of a knowledge base, knowledge acquisition facility, physical plant information base, inference system, and output system for HAZOP sheets and recommendations.

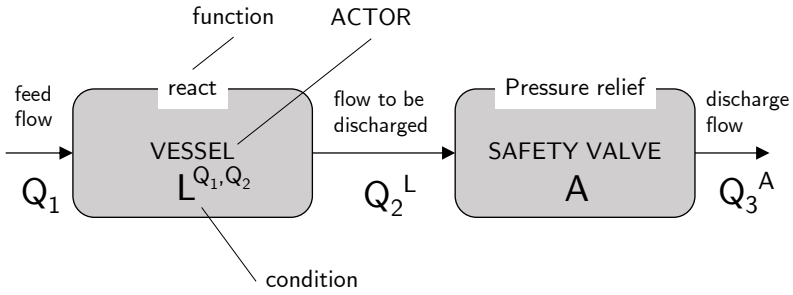


Figure 24: Exemplary D-higraph system (cf. Single et al., 2019b)

Information about the process plant configuration, such as piping, equipment, and instrumentation, must be configured manually and stored in the physical plant information base. The knowledge base contains knowledge about hazards and how they occur. The inference system evaluates rules defined in the knowledge base and the physical plant information. It was one of the first rule-based expert systems for HAZOP studies. The illustrations in the paper are conceptual in nature and do not show results (cf. Rootsaert and Harrington, 1992). To the best of knowledge, no available documents show the performance or achievable results of the approach. In addition to rule-based approaches, more powerful paradigms for knowledge representation and automatic inference are available today.

The research group around Schecker investigated the usage of knowledge-based systems to answer safety-related questions and developed the prototype HAZEXPERT (cf. Göring and Schecker, 1993; Kraus and Schecker, 2002). It is a rule-based approach that uses the expert system shell Nexpert Object. Another feature of the approach is extracting and using information directly from P&ID. On the other hand, no case study is publicly available to demonstrate the capabilities. Besides, no detailed accident or hazard model is provided, and safeguards and risk assessment are not considered.

Suh, Lee, Kang, et al. (1997) developed the automatic hazard analyzer (AHA) prototype (cf. Kang et al., 1999). It is built on the multi-model approach for reasoning and uses the expert system shell Gensym G2; knowledge is organized in the knowledge bases: equipment, organization, and material (Chittaro *et al.*, 1993). Hence, more expressive knowledge representation paradigms are available in more recent approaches. Furthermore, there are three hazard analysis algorithms for deviation, malfunction, and accident analysis. Based on that, the system builds an accident propagation path. The generated results must be interpreted and brought manually into the HAZOP worksheet form (cf. Section 7.3.3). Besides, risk assessment is not considered which is needed to propose adequate safeguards.

Lü and Wang (2007) proposed a computer-aided HAZOP tool and fault diagnosis methodology based on SDGs. The SDG model is built based on mathematical equations and P&ID information. SDGs model the plant layout and process and safety-related behavior of the plant. The *breadth-first-search* algorithm is used to search and traverse the SDG structure. Fault propagation paths are explored, while abnormal causes and adverse consequences are identified using inverse and forward inference methods. These strategies are used to improve the inference quality and avoid redundancies. The basis of the inference engine is the single fault origin assumption. The approach is applied to a synthetic ammonia production process. The authors state that the usage of SDGs for large-scale systems, redundant propagation paths, inherent quality ambiguities, and improvement of fault resolution (cf. Lü and Wang, 2007). The approach focuses on representing (propagation) paths, and external factors such as human error, site information, and systemic issues are not considered. Furthermore, risk assessment and safeguards are not considered.

Marx (2003) developed an expert system that analyses P&IDs based on recurring equipment configurations to detect potential hazards and suitable safeguards. The approach is integrated with CAD software, and conclusions are drawn using rules. The knowledge is stored in semantic

nets that lack formal semantics and limit the knowledge representation capability. The system is solely built on rule-based reasoning, which other reasoning paradigms can improve. Also, the approach did not aim to automate the HAZOP method but to identify potential hazards during the design process.

Bragatto et al. (2007) developed a prototype integrating hazard analysis methods with plant or product life cycle management (PLM) systems. The framework consists of the components: dictionary, plant information database, reasoning and analysis engine, and the knowledge repository. The dictionary is used to share and translate the used terminology, while the plant information database contains all the plant data and is coupled with the CAD software CATIA (cf. List, 2017). The reasoning engine is used to navigate through the documents and helps to conduct the HAZOP. The knowledge repository is based on three data models: a general hazard identification model, plant hazard identification model, and hazard analysis result model. The analysis is based on the P&ID, available as a CAD document within the PLM system. The project database can be accessed via an application programming interface (API). Each P&ID item, such as equipment, piping components, and instrumentation, is classified regarding its super-function, function, type, and functional parameters. The component-specific HAZOP knowledge must be entered manually, and can be reused, edited, and revised. Integrating HAZOP analysis with PLM is future-oriented because all process plant data can be accessed while equipment changes or revisions are noticed and considered. In the classical sense, the approach does not aim at automating HAZOP. The approach is meant for supporting experts during hazard analysis by allowing browsing through the project data, proposing causes and consequences. Hence, users are guided during the identification of scenarios. The automatic propagation of deviations causes and consequences through the process plant is not addressed. Propagations must be considered by the human expert when entering the HAZOP-specific details. Risk assessment is not considered, which is needed to propose adequate safeguards.

Schmidberger *et al.* (2009) proposed a knowledge-based approach for HAZOP studies based on a CAEX plant model. The idea of using a CAEX model of the plant is useful, as information regarding material flow and information flow (process control) between plant components can be used directly. The approach is rule-based for modeling relationships between deviations, causes, consequences, and plant component types. In the knowledge representation, causal relationships between cause and consequence and relationships to different modes of operation were not considered. The approach is missing a detailed characterization of equipment and its components to transfer and abstract scenarios. Finally, risk assessment of scenarios and safeguards were not considered.

Daramola et al. (2011) build the knowledge reuse-oriented safety analysis (KROSA) system, which combines: ontologies, case-based reasoning, and natural language processing (cf. Section 4.1.11). The approach tries to facilitate the reuse of knowledge from past HAZOP studies to support experts. Furthermore, the system was applied to three relatively simple systems: steam boiler, rail lock, and adaptive cruise control. The usage of natural language processing is a progressive approach to semi-automatically creating domain ontologies, which seems possible for simple technical systems (cf. Section 5.11). It is questionable whether the method can be applied to more complex chemical process plants. The integration of case-based reasoning and ontologies seems to be a promising approach. Nevertheless, the complexity of chemical process plants and the quality of the process-specific knowledge could be a limiting factor. Also, a component-based investigation is focused, and the propagation of scenarios through plant sections is not considered.

Wu et al. (2013) developed an ontology framework for the scenario-based hazard evaluation called scenario object model (SOM). It is an influence equation-based method to express structural information of scenarios. Within the approach, qualitative reasoning was used to draw conclusions. The idea of using ontologies for knowledge representation reflects the state-of-the-art knowledge storing capabilities. Besides a theoretical foundation, no details regarding the modeling of the

process system are provided. Also, no ontology structure used ontology language or proof that the proposed framework is suitable for the automatic creation of HAZOP worksheets is published.

Boonthum et al. (2014) introduced a method based on structural models. Thus, all relationships between process variables based on heat- and mass balance are represented using matrices. Based on HAZOP digraph models, a structural model is created. From that, a cause-and-effect matrix is derived for identifying consequences. The approach focuses on investigating relationships between process variables and not the automation of the HAZOP method. Therefore, aspects like reasoning about abnormal causes and consequences or an expressive knowledge representation are not considered. Scenarios that are not directly dependent on process variables cannot be captured. Besides, safeguards and risk assessment are not considered.

Park (2017) developed a data-based semi-automatic hazard identification (DAHAZID) method. An attempt is made to identify plausible scenarios based on control faults and equipment failure modes. Equipment and process data are extracted from Intergraph SmartPlant P&IDs via SQL queries. The formulation of SQL queries is similar to a rule-based approach, where the rules are contained in the queries. The application of the method leads to a pre-populated HAZOP worksheet. The idea of the approach was not to entirely automate the HAZOP process but to "stimulate people's imaginations as a checklist does but should leave space to insert uncommon cause and effect events emerging from the team deliberations" (cf. Park, 2017).

4.1.11 Ontologies in the field of process and plant safety

In the field of process and plant safety, researchers have recently developed ontologies to represent domain and application knowledge. Nevertheless, none of these approaches can be directly used for digitizing the HAZOP method. As part of this research, the first ontology aimed at digitizing the HAZOP method was developed and explored towards the end of 2018 and was published in a paper in 2019 (cf. Single *et al.*, 2019b). Further publications, including the design of the developed ontology, followed in 2020 (cf. Single *et al.*, 2020a; Single *et al.*, 2020b; Single *et al.*, 2020d).

The ontology developed in the KROSA prototype represents concepts and relationships of HAZOP scenarios which are not sufficient in the scope of this research (cf. Section 4.1.10). From the structure and relations in the KROSA HAZOP ontology can be concluded that scenarios were modeled in their entirety without considering the relationships between scenario components (cf. Section 3.2.4). By considering scenarios statically, the semi-automatic generation of ontologies is simplified. Apart from that, it makes an abstraction or transfer of scenarios exceedingly difficult since all scenarios have to be modeled explicitly without being able to conclude relationships between scenario components. Additionally, substance properties are not considered during the identification of scenarios.

Rodríguez and Laguía (2019) developed and proposed an ontology called OntoSafe for representing fundamental concepts, chemical process safety systems, industrial hygiene, safety standards, regulations and organizations, and emission and dispersion models. On a positive note, the complete ontology has been made publicly available. The ontology provides a good overview of the process safety engineering domain and provides explanations of terms and further resources. However, with the help of the available concepts and relations, HAZOP scenarios and their causal relationships cannot be fully represented. In addition, concepts and relations regarding safety-relevant substance properties and detailed characterization of plant components required for safety analyses are not provided. Hence, this ontology is insufficient to cope with the postulated goal of digitizing the HAZOP method (cf. Chapter 2).

Mao et al. (2020) worked on an application-oriented process safety knowledge graph for representing information embedded in HAZOP worksheets. The knowledge graph can be utilized for knowledge visualization, automatic information retrieval, and extraction of causal chains. Due to the different scope of the knowledge graph and its ontology, this approach is not sufficient to be

applied within this work due to rudimentary relations between concepts and inadequacy to fully represent causal relations within HAZOP typical scenarios. Also, the risk assessment of scenarios and safeguards are not considered within the ontology.

Gao et al. (2020) designed an application-oriented HAZOP ontology for petrochemical processes. The ontology is intended for realizing real-time analysis of safety states. The researcher made use of the Pellet reasoner and SWRL rules (cf. Sections 3.8.1 and 3.9.5). In this ontology, the relationships between HAZOP concepts and scenario components representing the causal sequence of events are not sufficient for the scope and extent of this work. The role of substance properties and their relationships to other scenario components are not considered. Also, risk assessment and safeguards are not considered within the ontological model.

4.2 Summary of the evaluation of the state of research

In this chapter, the technological state of the art and the methodological state of research were analyzed, processed, and evaluated. Overall, 32 approaches from the literature are analyzed in detail (cf. Table 8, Section 4.1, Single *et al.*, 2019a). Not all approaches aim to automate HAZOP studies per se but focus on specific aspects, such as:

- process and plant modeling,
- modeling of expert knowledge using different knowledge representation paradigms,
- directly use information from P&IDs or PFDs, e.g., topological information,
- consideration of the propagation of hazard scenarios throughout the process plant,
- reasoning techniques for handling typical and atypical scenarios.

Most research approaches use case studies or examples to demonstrate and evaluate the proposed methods. Sometimes the case studies are only conceptual, and the overall performance can not be derived. Identifying suitable safeguards and risk assessment remains an open issue in multiple approaches. Furthermore, not every causation model of the approaches is suitable to identify HAZOP typical scenarios. SDGs are used frequently to represent the process plant, process units, and their connections. More complex graph-based methodologies, such as MFM (cf. Section 4.1.8) or D-higraph (cf. Section 4.1.9), also contain specific knowledge, such as system behavior or involved process variables. The type of modeling technique of the process plant directly influences the type and scope of knowledge modeling and vice versa. For instance, SDGs could be combined with ontology-based models, which means that the essential information is contained in the ontologies. If more complex modeling techniques, such as MFM or D-higraph, are used, functional information is already embedded in the process and plant information. Taylor (2017) addresses the problem that overly complicated model paradigms can cause the modeling process for complex systems to take longer than the actual HAZOP study.

Based on the current state of research, ontologies are the means of choice for modeling interdisciplinary knowledge. The conceptualization of ontologies is complex and requires knowledge engineering expertise. The ontology language directly influences the reasoning capabilities of the application in which the ontology is used. OWL is a recognized and widely used ontology language recommended by the W3C as a standard modeling language. Various tools, libraries, and reasoners are compatible with the OWL language. Ontologies have been proposed for representing safety engineering knowledge, for equipment related incident investigation, and the modeling of dependencies between causes and consequences of hazards (Batres, 2004; Daramola et al., 2011; Mohammadfam et al., 2013; Bloomfield and Parisaca-Vargas, 2015; Mao et al., 2020; Gao et al., 2020). Nevertheless, none of the published ontology structures can be directly used to generate high-quality HAZOP worksheets automatically.

4.3 Gaps in Research

By systematically reviewing and evaluating the current state of research, the objectives formulated in Chapter 2 can be further concretized. No approach has achieved qualitatively equivalent results compared to human HAZOP conductors (cf. Chapter 2, (2)). Hazard identification methods are based on a causation model. The description and components of scenarios depend on this underlying causation model (cf. Figure 5). Thus, the scenario components and the underlying causation model must be reconsidered and investigated (cf. Chapter 2, (3)). According to the current state of research, it is unclear to what degree of abstraction and detail the process plant, equipment, substance, and site information must be modeled to identify scenarios reliably. Essential structural and functional relationships must be represented here without drastically increasing the modeling effort without any added value. The intended function of equipment and nodes must also be considered here, which has not been sufficiently considered in previous approaches.

Currently, no available ontology is suitable for use in knowledge-based systems for digitizing the HAZOP method in the chemical process industry. A knowledge representation is suitable if a software component automatically generates high-quality HAZOP worksheets for various process plants. Hence, the research question of how knowledge must be represented to be usable for the automation of HAZOP studies arises (cf. Chapter 2, (4)). Scenarios must be represented, considering the causation model behind them to achieve the research objective. A causation model is complete if the event sequence of scenarios can be represented unambiguously. Hence, a comprehensive ontology-based model for representing scenarios must be developed and investigated. Based on the achievable results, it has to be proven that ontologies are suitable for storing complex interdisciplinary knowledge and digitizing the HAZOP process.

Safeguards are only sporadically considered within past approaches, and even when they are considered, they are treated only as a secondary issue (cf. Section 4.1). Accordingly, this is a methodological gap because safeguards are not recognized in the required quality and completeness (cf. Chapter 2, (7)). Qualifying the risk of scenarios is required to determine the extent and the requirements for safeguards. The qualitative risk ranking of scenarios and their relationships to scenario components and safeguards has not been addressed sufficiently in past approaches.

McCoy et al. (2000) address that computer systems can produce too many implausible results that need to be filtered manually. This circumstance raises the research question of whether an ontology-based knowledge representation in conclusion with a specialized inference algorithm can prevent implausible inferences (cf. Chapter 2, (8)). Besides, automatically identified scenarios have to be checked for redundancy and plausibility. Furthermore, HAZOP method-specific aspects, such as different operation modes, must be investigated. Accordingly, case studies must assess whether different scenarios result from different operation modes and whether these are automatically recognized.

In addition to the equipment-based analysis, it must be investigated to what extent a systemic approach is necessary to identify hazards with sufficient reliability. The consideration of the propagation of events and the topology of the process plants affects the scenario identification. Propagation must be considered if it is the only way to identify valid causes of scenarios. Within the scope of this work, a propagation mechanism has to be developed, implemented, and tested to propagate events through plant components. The question is whether a computer system solves propagation differently from how HAZOP practitioners do it. Furthermore, socio-technical aspects must be discussed to highlight the possible applications and limitations. The question to be answered here is how knowledge-based systems can be integrated into the routines of HAZOP practitioners and to what extent computer systems influence the way experts work and their workflow.

Table 8: Summary of the literature review

Research approach	Year	Purpose/aim	Mechanism/Method/Technology	Open issues/implications
HAZOPEX | Section 4.1.1	1986-1999	Expert system for the identification of causes	Rule-based reasoning (expert system shell)	Focus is on causes of deviations, not consequences or safeguards
HAZOPTool | Section 4.1.1	1990	Expert system for generating HAZOP worksheets	Rule-based reasoning	No evaluation of the prototype available; safeguards not considered
Qualitative Hazard Identifier | Section 4.1.2	1995	Framework for hazard identification based on QDE	Qualitative differential equations;	Qualitative simulation models cannot be solely used for hazard identification; just for small plants
HAZEXPERT | Section 4.1.10	1993	Expert system for hazard identification based on P&ID data	Rule-based reasoning; P&ID information extraction; directed graph	No case study available; no accident model provided; safeguards not considered
(Auto)HAZID | Section 4.1.5	1993-2000	Expert system for generating HAZOP worksheets	SDG models; frame-based knowledge representation; rule-based reasoning	Limited ability to identify safeguards; the inference is mainly rule-based
AHA | Section 4.1.10	1997-1999	Expert system for deviation, malfunction, and hazard analysis	Rule-based reasoning; based on the *Multimodeling approach*	Knowledge representation can be improved; HAZOP worksheets must be derived from results
PHASuite (& predecessors) Section 4.1.4	1995-2005	Framework for scenario identification	Colored Petri-net model; digraph model; P&ID information; case-based reasoning	Knowledge representation in need of improvement; insufficient consideration of safeguards
ExpHAZOP+ (& predecessors) | Section 4.1.3	1997-2009	Expert system for hazard identification	Frame structure and rule networks; graphical editor for modeling of flow diagram	The accident model can be improved; rule networks are difficult to maintain; no safeguards are considered
SafeCAD | Section 4.1.10	2003	P&ID based hazard identification	Rule-based reasoning; P&ID information	The aim is not to automate HAZOP; limited reasoning and knowledge representation capabilities
LDGHAZOP | Section 4.1.7	2008	Framework for hazard identification based on P&ID	Layered digraph (LDG) model; P&ID information; deviation propagation	Causes and consequences are not semantically connected; no safeguards are considered
HAZOPSuite (PetroHAZOP) | Section 4.1.6	2009	Ontology-based scenario identification	Ontologies; case-based reasoning	No ontological model was provided; the level of abstraction of equipment is insufficient
MFM HAZOP assistant | Section 4.1.8	2010	Framework for scenario identification	Multilevel Flow Modeling; rule-based reasoning (JESS expert system shell)	Focus is on the identification of root causes of faulty scenarios; no consequences or safeguards
KROSA | Section 4.1.10	2011	Ontology-based scenario identification	Ontologies; case-based reasoning	Focus is on component-based evaluation; propagation not considered; ontological model in need of improvement
D-higraph HAZOP assistant | Section 4.1.9	2012	Framework for graph-based scenario identification via causal trees	D-higraph; rule-based reasoning engine	It aims to identify causal trees of deviations; currently, no consequences or safeguards are considered
CAH (SOM) | Section 4.1.10	2012	Ontology-based scenario identification	influence equation-based method	No ontological model provided
HAZOP structural models | Section 4.1.10	2014	Framework for identifying cause and effect relationships	Digraph model; cause-effect matrix	Focus is on cause/consequences derived from general heat and mass balance relations; no safeguards
DAHAZID | Section 4.1.10	2017	P&ID based hazard identification	P&ID information; SQL queries; rule-based reasoning	Prepopulated worksheets serve as a guideline for a team; specific cause-consequence relationships are not inferred automatically

5. Development of a procedure for computer-aided HAZOPs

After establishing the state of knowledge and the research gaps, the development and investigation of a knowledge-based system will be presented. The development process is divided into five steps:

1. Derivation and proposal of an ontology-based causation model for representing HAZOP scenarios. In Section 5.1, principal considerations, the formulation of preliminary competency questions, and resulting relationships between core concepts are outlined.
2. The design, implementation, and investigation of the approach developed in this work are described in this chapter. It involves a graph-based and object-oriented representation of plants, equipment, substances, and site information (cf. Section 5.4-5.6). Section 5.7 presents the modeling of complete plant sections.
3. Based on the findings from 1. and 2, the developed ontology, domain-specific content, extent, and formalization are described in Sections 5.8 to 5.10. In Section 5.11, a method to generate ontologies semi-automatically is proposed.
4. The development and implementation of the inference algorithm as part of a software component are described in Sections 5.12 and 5.13. It exploits the ontology and utilizes reasoning mechanisms to analyze plant sections, identify scenarios, conduct risk assessments, and propose safeguards.
5. Section 5.15 lays out strategies to propagate deviations and events through plant sections under consideration of the topology. Also, the developed algorithms are described.

Finally, the application workflow of the developed method and the knowledge-based system is presented in Section 5.15, and Section 5.17 summarizes the approach developed and the lessons learned.

5.1 Further requirements

In addition to the objectives and requirements from Chapter 2, the assessment of the state of knowledge and the research gaps result in further requirements:

- State-of-the-art ontology-based knowledge representation and reasoning paradigms shall be considered. The knowledge representation must be limited to the minimum necessary to generate high-quality HAZOP worksheets automatically. The criteria formulated by Gruber will be considered in the development of the ontology (cf. Section 3.8).
- During the conceptualization of the ontology, the design decisions will be presented to enable the developed approach's comprehensibility, reusability, transferability, and extensibility.
- When representing plants, it is crucial to identify the level of detail and abstraction required to infer structural and functional similarities between the equipment. It should be possible to identify and transfer scenarios unambiguously. Hence, different levels of complexity should be investigated.
- The propagation of events through the process plants must be considered. In this context, the influence of the plant-specific topology on the inference strategy should be examined.
- Due to the nearly unlimited number of possible HAZOP scenarios within the chemical process industry, it is essential to avoid modeling scenarios statically in their entirety. It is necessary to assemble scenarios dynamically based on their components to ensure the transferability of scenarios.

- Ambient conditions and environmental dangers must be considered to ensure the plausibility of the causes of scenarios.
- Different modes of operation and intended functions of the equipment shall be considered.
- Due to the versatility and complexity of plant types and processes within the chemical process industry, not all possible HAZOP scenarios can be modeled for all theoretically possible process engineering apparatuses, processes, and substances. This circumstance leads to a *content completeness problem* because not every concept of interest can be represented (cf. Elkin et al., 1999). Therefore, the developed method has to be especially generically applicable and transferable to be extended.
- The absolute completeness of a knowledge representation can never be achieved but only incrementally approached. Hence, quasi domain completeness can be approached for strictly limited types of equipment and must be successively enhanced to cover more and more equipment. Kidam and Hurme (2013) identified the equipment most often involved in accidents: piping system, storage tank, reactor, heat transfer equipment, process vessel, and separation equipment. Hence, this research work focuses on particularly relevant equipment to narrow the scope of application meaningfully. As plant components consist of piping systems and connect plant sections, they are considered within this work. Additionally, different types of tanks are considered. In a broader sense, reactors and separation equipment represent special pressure vessels.

5.2 Overview of the approach and principal considerations

For automating the HAZOP method, the steps performed by HAZOP practitioners must be digitized and transferred into a computer system. Therefore, the HAZOP process is modeled in an engineering-oriented manner and described using computer science methods. Figure 25 shows a schematic overview of the components of the developed method. First, the plant section to be analyzed must be transformed into a graph-based representation based on a P&ID or PFD. In addition, substance-specific and site information must be provided and integrated into the graph in the form of objects. The required expert knowledge is acquired, modeled, formalized, and made available in the form of an ontology. Based on the graph-based representation of the plant, process, environment, and substances and the ontology, the developed prototype of a knowledge-based system identifies scenarios, assesses the risk of the scenarios, and proposes appropriate safeguards. Finally, the results are automatically summarized in HAZOP typical tables (worksheets).

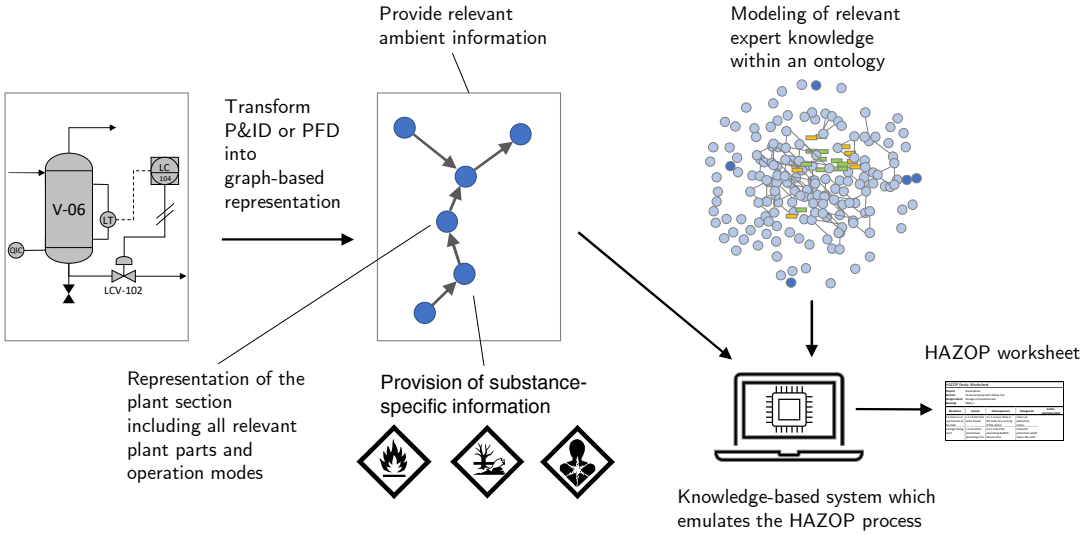


Figure 25: Overview of the method developed for the automation of HAZOPs

5.2.1 Overview of the Ontology development process

The ontology development process is divided into five steps in this work (cf. Section 3.8): specification, conceptualization, formalization, application, and evaluation. At the beginning of the ontology development process, the scope of the ontology, its role, and use cases within the software application must be defined (specification). An ontology will be developed to represent causal relationships within scenarios. It will be used in the context of scenario identification and risk assessment within HAZOP studies of knowledge-based systems to draw inferences. Due to the breadth of knowledge from different disciplines, such as process engineering, process control, plant engineering, and safety engineering, the knowledge to be represented is interdisciplinary. The developed ontology must be investigated to identify the required level of abstraction and detail and the use within a knowledge-based system. In addition, interfaces between knowledge concepts must be identified.

There is no ontology available within the scope and requirements of this research work that can be reused (cf. Section 4.3). For individual aspects of the classification of plant components, the OntoCAPE ontology serves as a first orientation (cf. Figure 36). Initially, the level of detail and abstraction required to achieve the goal of computer-aiding HAZOP studies is unknown. Achievable results depend not only on the ontology but also on the design and implementation of an inference algorithm that draws conclusions based on the ontology. Since neither ontology nor inference algorithm are available a priori, preliminary approaches and conclusions can only be evaluated during development.

Some guidelines describe the schematic approach for developing ontologies (cf. Noy and McGuinness, n.d.). However, there is no standardized approach to modeling domain knowledge because it depends on the application, the domain, the scope, the granularity, the domain experts involved in providing knowledge. Accordingly, the initial degree of freedom is very high. The knowledge domain required to automate the HAZOP method is not constrained at the beginning of the development, and required concepts are not known a priori. Due to the vastness of the knowledge domain required for automating the HAZOP method, it is essential to avoid information overload. First, the knowledge domain has to be delimited based on the scope of the ontology. The domain can be roughly delimited by formulating competency questions to be answered with the help of the ontology. Afterward, knowledge acquisition and conceptualization can take place.

Domain expertise and expertise in knowledge engineering are required to design and conceptualize an appropriate ontology that can be used within the context of a knowledge-based system. During conceptualization, associations between concepts are established based on knowledge acquisition. It includes modeling the causal chains of scenarios based on the aggregated domain knowledge. The consistency of different knowledge sources must be considered during knowledge acquisition, and inconsistencies must be compensated using their own or external domain knowledge. Knowledge acquisition, conceptualization, and formalization are conducted iteratively as the concepts are extended, refined, and revised in the ontology life cycle. The formalization of the conceptualized ontology requires awareness regarding formal logic and set theory. Based on different conceptualization, formalization, and application activities, experience with the ontology language OWL 2 and the use of the HermiT and Pellet reasoner was gained in this work.

In addition to the vast domain, unknown granularity, and unknown degree of abstraction, it is unclear which expressiveness the ontology language must have at the beginning of the development process. The concepts required for the fulfillment of the objective can be represented unambiguously. Therefore, an evolutionary development approach was used in this work. The feature set is refined and redefined during the development because, after completing individual components and examining these, new requirements arise, and old requirements change. Therefore, prototypes of different aspects have been developed to investigate ideas, structures, and required relations between concepts. Afterward, these prototypes have been evaluated, rejected, accepted,

and refined during the ontology development. Since ontologies only provide results in combination with an inference algorithm, an algorithm had to be designed and developed simultaneously as the ontology. The development, exploitation, and evaluation of the ontology and the inference algorithm are conducted iteratively. With new findings, the set of features is refined and redefined during the development of features, e.g., considering risk assessment.

The initial focus within the ontology development process was on representing scenarios based on the scenario components. The modeling of equipment and substances was initially deliberately kept simple. The restriction to the essential without a too coarse level of detail is crucial for drawing the correct conclusions. As development progressed, concepts were abstracted further and further to ensure transferability where possible, and the level of detail of equipment and substances was significantly increased. These enhancements also include data properties for representing numerical substance properties. The stages of ontology development are presented after presenting the contents and concepts of the ontology in Section 5.10.

The evaluation of the ontology is conducted application-based, which means the generated results (HAZOP worksheets) are analyzed to evaluate the concept coverage and quality of the ontology and the sufficiency of the reasoning algorithm. Ergo, an isolated evaluation of the ontology is not feasible in the scope of this work. Additionally, ontology metrics are used for evaluating and comparing developed ontologies (cf. Section 7.2.5).

5.2.2 Narrowing the knowledge domain

The required resources, documents, and expert knowledge gathered and elaborated within HAZOP studies must be analyzed to capture and constrain the knowledge domain to be modeled. In addition, the results of the studies in the form of worksheets must be examined. These worksheets are composed of prerequisite process safety information and the implicit knowledge (experience-based knowledge, cf. Section 3.4) of the study participants. The participants possess knowledge of processes and operations, process control and instrumentation, safety engineering, and the involved substances. Figure 26 shows related knowledge fragments of a HAZOP worksheet and linkages to supplementary information sources. The scenarios in tables correspond to explicit knowledge concluded from supplementary information sources (R&ID, SDS, process safety description, equipment data sheet) and the implicit knowledge of the participants (cf. Section 3.5). *Information* without context corresponds to *data*, e.g., the reference drawing number 'P&ID: D-007' (cf. Figure 26). The header of a HAZOP worksheet contains essential information, such as a description of the plant section under consideration (node), which is derived from the P&ID. Additionally, a (short) description of the design intention provides information about the process design of the node which is composed of the process description and the explicit and implicit knowledge of the plant engineers, plant managers, and operating staff.

All worksheet layouts have in common that the columns and the header are related (cf. Figure 26). There are semantic relationships between deviations, causes, and consequences. Besides, scenario components, such as cause and consequence, may depend on the intended functions and design parameters. Also, there are relationships between the likelihood and cause and between severity and consequence. These relationships suggest that there is a causation model behind the structure of the scenarios. Besides, plant and equipment-specific and substance information are included in the worksheet cells (cf. Figure 26). HAZOP practitioners extract equipment relevant information from available P&IDs, PFDs, and information provided by the operator. A computer system requires an adequate representation of the plant and its components. Additionally, information about the involved substances, modes of operation, and site information are required and must also be considered during the ontology development.

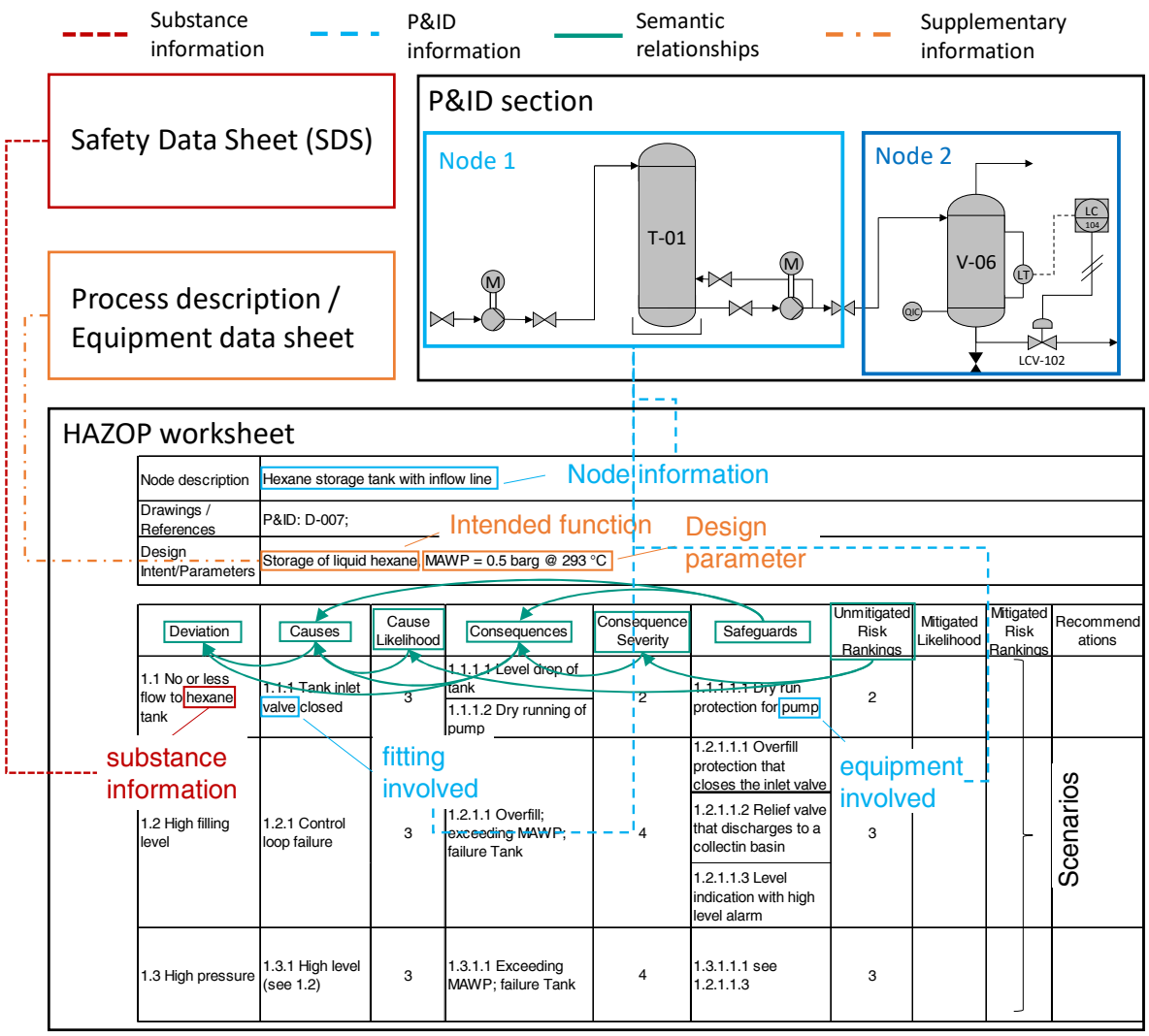


Figure 26: Related knowledge fragments within HAZOP worksheets

5.2.3 Investigation of causation models and core HAZOP concepts

The structure and components of scenarios are analyzed as part of the ontological model development. Therefore, fundamental assumptions behind scenarios and their causal relationships are captured and represented. A combination of a top-down and a bottom-up approach was chosen within this work. On the one hand, an underlying model is sought for describing scenarios. Within a bottom-up approach, industry-typical HAZOP worksheets are analyzed to capture the rules that apply to the represented scenarios. Within the bottom-up perspective, individual scenarios and their components were analyzed to identify the governing theoretical framework behind them. Initial guidance on causes for overpressure scenarios within a system can be found in ISO 4126-10 in Annex A. Furthermore, in the course of knowledge acquisition, the following activities were carried out:

- participation in industrial HAZOP studies,
- examination of industry typical HAZOP worksheets (cf. Section 3.2.7; Marhavilas et al., 2020),
- the analysis of weaknesses of the HAZOP method (cf. Section 3.3),
- examination of the used terminology (cf. Section 3.2.1),
- investigation of causation models (cf. Section 3.2.3, Section 5.2.3),
- investigation of failure scenarios (cf. CCPS, 1998; Nolan, 2015),

- investigation of knowledge concepts, such as fault trees, from the literature (cf. Fuentes-Bargues et al., 2017),
- discussions with domain experts regarding the usage of terms, concepts, and the context they are used in,
- discussions with domain experts to gain a systemic understanding of specific equipment such as pumps and storage tanks and scenarios.

After aggregating domain knowledge, the ontology is conceptualized, which includes identifying important terms, deriving concepts, establishing associations between concepts, and determining the granularity of the associations. In the process, attention was paid to semantically consistent modeling of fundamental relationships between contexts to describe the meaning of the concepts unambiguously. The relevance of modeling the semantic context of concepts within the scope of automating HAZOP studies was described by Single *et al.* (2020b). Additionally, the underlying causation model of typical HAZOP scenarios is considered during conceptualization. As described in Section 3.2.3, scenarios are based on a causation model representing an event sequence (cf. Figure 31). These are usually linear models within the classic HAZOP method (cf. Figure 27, I, II). The starting point is the typical separation of scenario components into **Cause → Deviation → Consequence** (cf. Figure 27, I). Within conventional HAZOPs, study participants identify, interpret, abstract, and describe causes and consequences to describe scenarios. Within this model representation, the cause represents the initiating event (cf. Figure 31), from which the process deviation arises and results in a consequence (loss event). Following the definition from Section 3.2.1, all conceivable outcomes of events are presented in industry-typical HAZOP tables in the consequence column, such as equipment failures, loss events, production-related consequences. The following consequences should serve as examples:

- 'Environmental contamination and possible explosion' (cf. ISO 61882:2016, pp. 33),
- 'Possible reduction in yield' (cf. ISO 61882:2016, pp. 31),
- 'Inadequate net positive suction head' (cf. ISO 61882:2016, pp. 31),
- 'Loss of production' (cf. Crawley and Tyler, 2015b, pp. 115),
- 'Thermal expansion of the pipeline' (cf. Crawley and Tyler, 2015b, pp. 115),
- 'Overheating with possible reaction runaway' (cf. Crawley and Tyler, 2015b, pp. 134).

Depending on the way of expression, problems can arise in the comprehensibility of scenarios. Causal relationships within the described scenarios may be interpreted differently based on the context. During the HAZOP study, the practitioners must establish the relationships and event sequences behind scenarios based on professional experience. Because practitioners have different ideas regarding the causal model of scenarios, scenario descriptions differ. For a knowledge-based system to perform scenario identification, such relationships between scenario components must be explicitly distinguished and modeled within the knowledge representation.

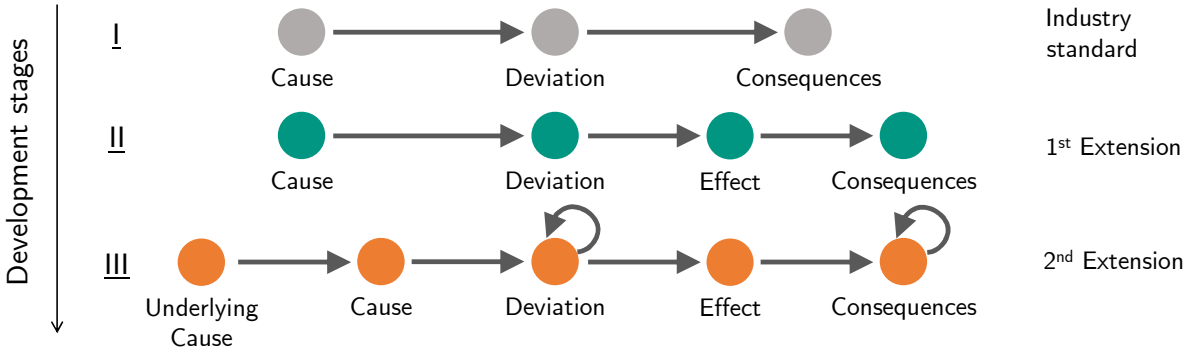


Figure 27: Causal sequence of events of scenarios

For example, in industrial practice, it is common to assign deviations to the category **Consequence**, such as the deviation 'high level in a closed vessel' leads to the consequence 'high pressure' (cf. Figure 26). A knowledge-based system cannot draw this conclusion due to semantic differences. Assigning a deviation to a consequence leads to inconsistencies because a concept from the category **Deviation** cannot be assigned to the category **Consequence**. Furthermore, without clear distinctions and classification of the scenario components, knowledge-based systems cannot readily interpret textual descriptions, leading to potentially incorrect conclusions. A fine-grained description of the components of the scenarios, considering the relationships of the components of the scenarios, helps to improve the reasoning capabilities of knowledge-based systems. A more precise delineation of concepts and introducing an event between **Deviation** and **Consequence,** namely **Effect**, simplifies scenarios' comprehensibility (cf. Figure 27, II). It is possible to describe intermediate events between deviation and consequences using the effect category (cf. Figure 28). Table 9 shows examples of effects and consequences and the corresponding deviations. For example, a semantic distinction is made between the **Effect** *pool formation* or *gas dispersion* and the **Consequence** *loss of primary containment* because further analyses such as dispersion or pool evaporation calculation differ significantly (cf. Table 9).

Additionally, consequences can lead to subsequent consequences that form a consequence sequence (cf. Figure 27, III). For instance, a compressor breakdown may lead to production downtime of a plant or plant section. Deviations can also lead to subsequent deviations. For instance, a 'low temperature' deviation in a pipe leads to a 'no flow' deviation if the liquid freezes. The term cause and thus the concept **Cause** covers a wide range of different types of causes, such as equipment specific, systemic, management, and organizational causes.

The **UnderlyingCause** concept was introduced to distinguish different causes at different levels, such as systemic and organizational causes and causes arising from latent human errors. For example, insufficient training, pre-start-up checks not executed, or solar radiation can be assigned to underlying causes. The additional scenario component makes it possible to distinguish causes at different hierarchical levels. Based on these considerations, Figure 5 can be extended to include the concepts **Effect** (intermediate event) and **UnderlyingCause** (underlying event), resulting in Figure 28. An underlying cause comes before a cause in the event sequence, while an effect follows a deviation and is the predecessor of a consequence. The *scenario components* and the *events* are disjoint (cf. Figure 16). Scenarios with no immediate effect or consequence are referred to as *partial scenarios* in this work. Partial scenarios are described in detail in Section 5.14.

Table 9: Example distinctions between effects and consequences

Deviation	Effect	Consequence
High pressure	Pressure exceeding design pressure; fracture	Loss of primary containment
Elsewhere flow	Pool formation	Loss of primary containment
Low level	Emptying of container	Production downtime
Other than composition	Liquid slugging	Compressor breakdown

Table 10: Example distinctions between underlying causes and causes

UnderlyingCause	Cause	Deviation
Operational error	Valve wrongly closed	No flow
Human error	Confusion of chemicals	Other than composition
Failure control loop	Cooling failure	High temperature

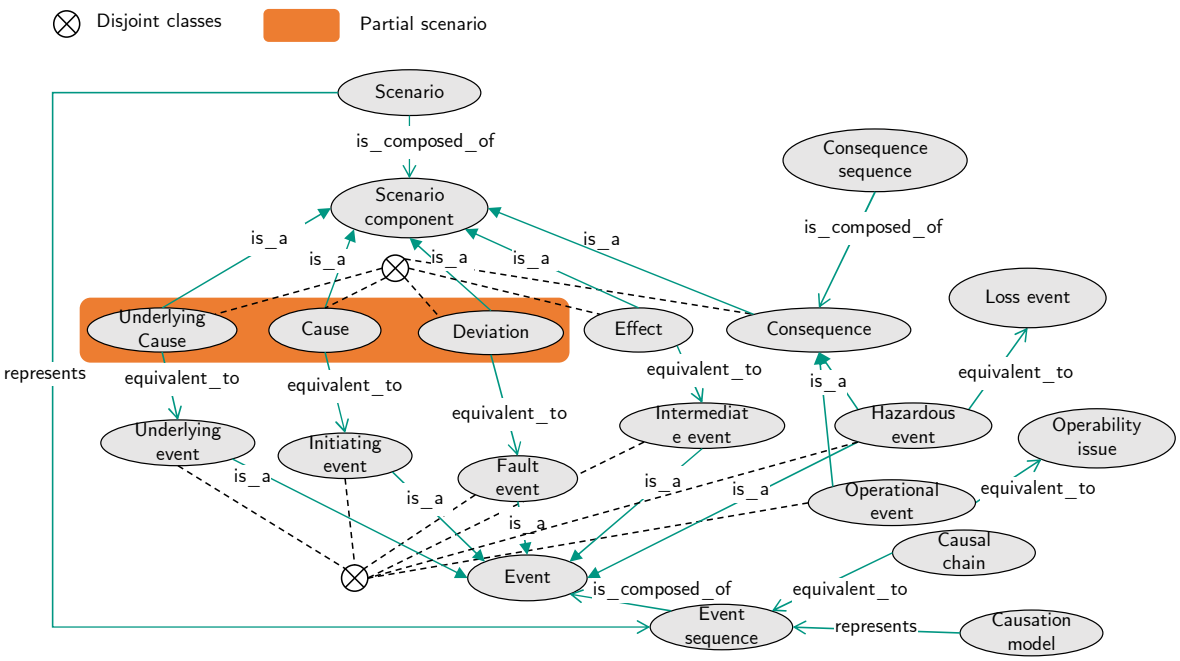


Figure 28: Enhanced concept map of working concepts

5.2.4 Competency questions and domain-specific requirements

Competency questions represent questions that software components or users of the ontology want to be answered. Competency questions can be answered by browsing the ontology and exploring the content, for example, using the ontology editor Protégé. Additionally, the ontology querying mechanism can be used. With the help of competency questions, the scope of the ontology can be formulated, and the knowledge domain can be narrowed down. They serve as a starting point for the conceptualization of the ontology. The use of competency questions within the scope of process safety engineering ontologies has been described previously (cf. Single *et al.*, 2020ds; Single *et al.*, 2020a). The application and how the ontology is used determine the competency questions. Supposed the ontology is explored manually with the help of an ontology editor, a different spectrum of competency questions results than if a software component performs specific queries against the ontology. Competency questions are not exhaustive, and the ontology can answer significantly more questions than stated within the competency questions. Hence, ontology questions represent a minimum requirement. Based on the related knowledge fragments within HAZOP worksheets (cf. Figure 26), the HAZOP terminology (cf. Section 3.2.1), and the considerations regarding the causation model of scenarios (cf. Section 5.2.3), the following competency questions are formulated:

- CQ 1: What are the potential causes of a specific deviation?
- CQ 2: Are there superordinate (underlying) causes?
- CQ 3: What are the potential consequences of a specific deviation?
- CQ 4: How are the causes and consequences related?
- CQ 5: Are there any subsequent consequences of consequences?
- CQ 6: Are there typical safeguards related to the deviation?
- CQ 7: What are the possible safeguards to prevent a cause?
- CQ 8: What are the possible safeguards to mitigate a consequence?
- CQ 9: What equipment is involved in the cause?
- CQ 10: Which equipment is involved in the consequence?
- CQ 11: Which substance properties engage in the consequence?

- CQ 12: Which substance properties engage in the cause?
- CQ 13: What site information engages in the cause?
- CQ 14: What site information influence the consequence?
- CQ 15: Into which risk category can the scenario be assigned?

The ontology to be developed will provide a knowledge-based system with HAZOP domain knowledge. Primarily, the ontology is intended to represent causal relationships within HAZOP scenarios. Based on the ontology-based framework, application-specific scenarios shall be represented. Besides, information for modeling process plant sections, substances, and site information should be provided by the ontology. The purpose of use gives rise to the following domain-specific requirements for the ontology:

- It must be possible to distinguish between different types of causes because the initiating event of a scenario must be reliably determined considering the causal sequence,
- A causal model must represent the relationships between the scenario components deviation, cause, consequence,
- Consequences often have subsequent consequences (consequence A → consequence B); consequence sequences must be representable,
- Appropriate safeguards and organizational measures must be identifiable,
- Substance, equipment, and site-specific relationships are required for unambiguously modeling scenario components,
- To avoid static non-transferable modeling of scenarios, the event sequence of scenarios must be comprehensible. Hence, scenarios must be dynamically composed of scenario components.
- It must be possible to determine the scenario-specific risk of scenarios and identify appropriate safeguards. Hence, relationships between likelihood, severity, risk, and the scenario components should be modeled.
- The ontology structure should be optimized for automatically identifying scenarios and risk assessment.

5.2.5 Naming convention

Within the conceptualization and formalization, a consistent naming convention is followed:

- Relationships between concepts (object/data properties) are denoted in an underlined font in lower camel case notation, e.g., deviationHasGuideword,
- Schema-oriented ontology class definitions are denoted in bold font upper camel case notation, e.g., **HighPressure**,
- Class definitions of the knowledge base are denoted in double quotation marks in upper camel case notation, e.g., "AbnormalEvaporation",
- Class names are expressed as singular nouns, e.g., consequence instead of consequences,
- Abbreviations in class names are avoided, e.g., maximum instead of max.

5.2.6 Modularization of the ontology

The final version of the ontology is divided into the modules *HazopCoreConcept*, *RiskAssessment, Equipment, Substance, SiteInformation,* and *BoundaryCondition* (cf. Figure 29). Modularization makes it possible to separate domain-specific details, manage dependencies between modules, and improve the maintainability of the modules. Since the ontology modules are created in different files, the import mechanism must also be considered to avoid circular dependencies. The modules *Equipment*, *Substance*, *SiteInformation*, and *BoundaryCondition* are independent and

interchangeable, while the *HazopCoreConcept* and *RiskAssessment* modules depend on all other modules.

5.3 Conceptualization of the HazopCoreConcept

During conceptualization, concepts describing scenario components and their semantic relationships to each other are identified and modeled. First, the most significant concepts and relationships are identified within the knowledge domain to conclude further. The result is a set of related terms and a class hierarchy, which can be visualized in a concept map (cf. Figure 28). The *HazopCoreConcept* module represents causal relationships between scenario components and safeguards (cf. Figure 26). After presenting essential relationships in this chapter, more advanced relationships and examples are presented in Sections 5.8 and 5.9. First, the deviation concept and its relationships are illustrated:

$$\text{Deviation} = \text{Deviation}(\text{Guideword}, \text{Parameter}). \tag{1}$$

A **Deviation** usually consists of a **Guideword** and **Parameter**, expressed through the relationships deviationHasGuideword and deviationHasParameter. In Figure 30, the description in the brackets indicates how the concepts are related. Referring to the triple notation (cf. Figure 13), the *domain* entry represents the *subject*, while the *range* entry represents the *object*. The relationship itself is the *predicate*. Analogously, the relationships between all required concepts are established within this work. The concept map in Figure 31 visualizes all HAZOP core concepts and their relationships. **UnderlyingCause**s directly depend on causes but cannot be linked to a specific deviation since underlying causes refer to higher-level causes such as systemic, organizational, and human errors. Therefore, underlying causes can be easily transferred between scenarios. Thus, **Cause**s are related to underlying causes using the isUnderlyingCauseOfCause relation. The cause concept is related to the deviation concept via the object property isCauseOfDeviation.

The **Effect** concept (cf. Figure 27) is related to the deviation concept via isEffectOfDeviation since effects are derived based on deviations (cf. Figure 31). Besides, the effect concept is related to the causes via object property effectImpliedByCause. In earlier development stages, the relationship between effect and cause was not considered (cf. Single et al., 2019c). In terms of the standard industry approach, this means that consequences are identified based on causes. Hence, effects are identified based on causes in this work. Furthermore, consequences are derived from effects because they follow effects within the causal chain of the scenario (cf. Figure 27, II).

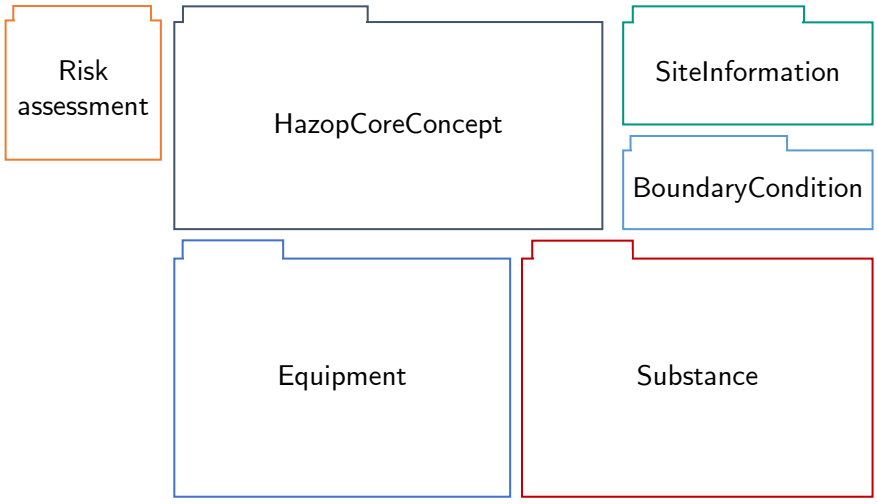


Figure 29: Modularization of the ontology

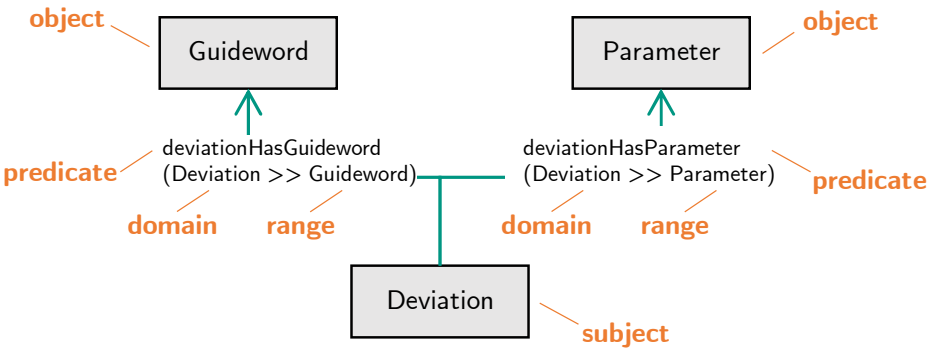


Figure 30: Relationships for the generation of deviations

Consequences are connected via the relation isConsequenceOfEffect to effects. Additionally, consequences are related to deviations via the isConsequenceOfDeviation relation to further specify the intention behind the concept. In rare cases, consequences are also related to causes via consequenceImpliedByCause. This relationship allows an indirect connection without the necessity of specific effects to represent consequences. The use of the object properties isConsequenceOfDeviation and consequenceImpliedByCause proved to be relevant to represent consequence sequences unambiguously.

Following the definition from Section 3.2.1, all protective and safety measures, such as

- Preventive safeguards, e.g., automatic action, operator supervision, pressure relief valve,
- Mitigative safeguards, e.g., deluge system, dike,
- Operative measures, e.g., Basic Process Control System (BPCS) measure, alarms, controls, and operating procedures

from the Independent Protection Layer (IPL) are subsumed under the concept **Safeguard** within this work. The consideration of different types of safeguards agrees with the industrial practice of presenting different safeguards together in the safeguard column of HAZOP worksheets (cf. Baybutt (2012); Marhavilas *et al.* (2020); IEC 61882:2016). **Safeguards** are intended to prevent (underlying) causes and effects and mitigate consequences (cf. Section 3.2.1). The safeguard concept is connected to the cause concept via **safeguardPreventsCause** and safeguardPreventsUnderlyingCause and effects via safeguardPreventsEffect. Additionally, safeguards are related to the consequence concept via safeguardMitigatesConsequence. Besides, the safeguard concept is connected to deviation via safeguardOfDeviation because appropriate measures can be determined based on the equipment and deviation (cf. Section 5.8.5).

The competence questions defined in Section 5.2.4 are indicated in Figure 31 alongside the relationships. The developed schema-oriented ontology containing the core concepts is reused to model specific application-oriented concepts. Labels are used to describe the context of the core concepts via annotations to clarify the design criterion (cf. Section 3.8; Section 26; Appendix Listing G-1; Appendix Listing G-2) HazopCoreConcept, relationships with equipment, substance, or site-specific concepts are not yet represented. These must also be considered to fully model scenarios described in the following.

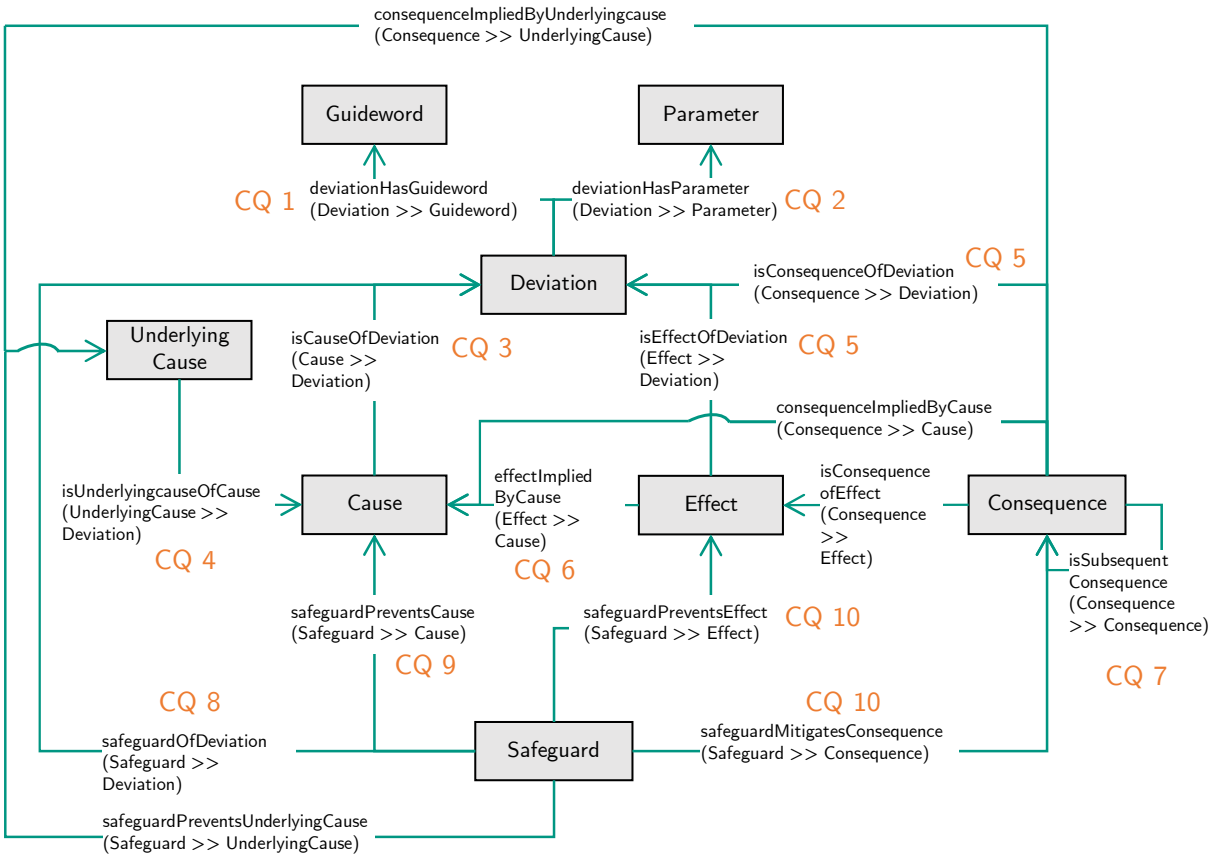


Figure 31: Concept map of the *HazopCoreConcept*

5.4 Substance entity

In addition to the core concepts, substance information is required for scenario identification. Safety data sheets (SDS), formerly known as material safety data sheets (MSDS), contain safety-relevant substance information. The content and structure of an SDS are defined within the REACH (Registration, Evaluation, Authorization, and Restriction of Chemicals) regulation of the European Union. Within the SDS, the *Globally Harmonized System of Classification, Labelling, and Packaging of Chemicals* (GHS) (cf. United Nations, 2019) system is used for the uniform classification and categorization of hazards.

Based on the SDS, the structure of a substance object was developed, which represents substance information. The entire content of the SDS is not required for describing scenarios and identifying potential hazards and operability issues, but the information required for modeling concepts is selected. This procedure also corresponds to the design criterion of *minimal ontological commitment* (cf. Gruber, 1995). If necessary, the object can be extended. A concept map of the substance object and its relationships is shown in Figure 33. A substance is uniquely identified by its chemical abstracts service (CAS) number, which is done within the substance object using the data property hasCasNumber. Furthermore, the intended state of aggregation of a substance is required for representing causes and consequences adequately. This relationship is expressed with the object property hasStateOfAggregation (cf. **StateOfAggregation**, Figure 33).

Based on the GHS system, hazard classes are represented within the substance object, divided into **PhysicalHazard**, **HealthHazard**, and **EnvironmentalHazard** (cf. **HazardClass**, Figure 33). The category is directly considered in naming the hazard classes, e.g., **FlammableLiquidCategory1**. The hazardous properties of substances define potential consequences and are

required to distinguish them. For instance, a loss of containment of a substance with the hazardous property **AcuteToxicity** would indicate a toxic release. The hazard class **FlammableLiquid** of a substance, in conjunction with oxygen available and an ignition source present, would indicate a fire. Besides, the hazard classes are needed to identify appropriate safeguards, e.g., if a storage tank contains a flammable substance, a flame arrester may be recommended by national standards.

Furthermore, the stability and reactivity of substances play a decisive role in the definition of concepts, which is subsumed within the class **StabilityReactivityInformation** and can be related to the substance via the object property hasStabilityReactivityInformation (cf. Figure 33). For instance, this may include the tendency to a violent reaction or the formation of an explosive mixture in contact with air (oxygen). In addition, quantitative values of physical and chemical properties are also needed for specific class definitions (cf. **PhysicalChemicalProperties**, Figure 33). For instance, safeguards can depend on the flashpoint of substances. The solidification point of the substance may also be required to identify specific concepts. For example, when the hazard of freezing water in pipelines is evaluated. Accordingly, the flashpoint, solidification point, vapor pressure, upper and lower explosion limit, and autoignition temperature are considered within the substance model using the data properties hasFlashpoint, hasSolidificationPoint, hasVaporPressure, hasUpperExplosionLimit, hasLowerExplosionLimit, and hasAutoIgnitionTemperature (cf. Figure 33). These data properties may be extended if needed for modeling concepts. Independent of typical SDS data, the substance object also contains information regarding the process-specific task of the substance. Initially, the equipment entity's intended function was assumed to be sufficient for modeling further concepts (cf. Single et al., 2020d).

However, it turned out that the specific task of substances (e.g., inert gas, refrigerant, process medium) is needed to model specific concepts unambiguously. For instance, the NoFlow deviation within a StorageTankEntity may lead to the Effect "EmptyingOfContainer" because no liquid is supplied (cf. Figure 32, a). If this storage tank also involves nitrogen blanketing, this effect would also be inferred for the inert gas, which is factually incorrect. A factually incorrect but semantically correct inference can be avoided by adding the substances task, as shown in the Venn diagrams in Figure 32. Accordingly, the substance task constitutes a part of the description of the intended function.

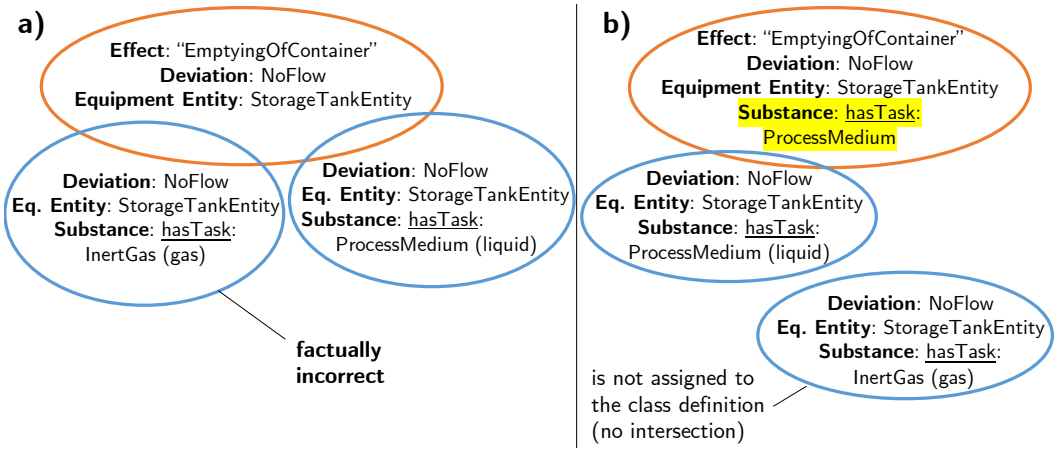


Figure 32: a) Overlapping of class definitions without considering the substance task
b) Unambiguous class definitions under consideration of substance tasks

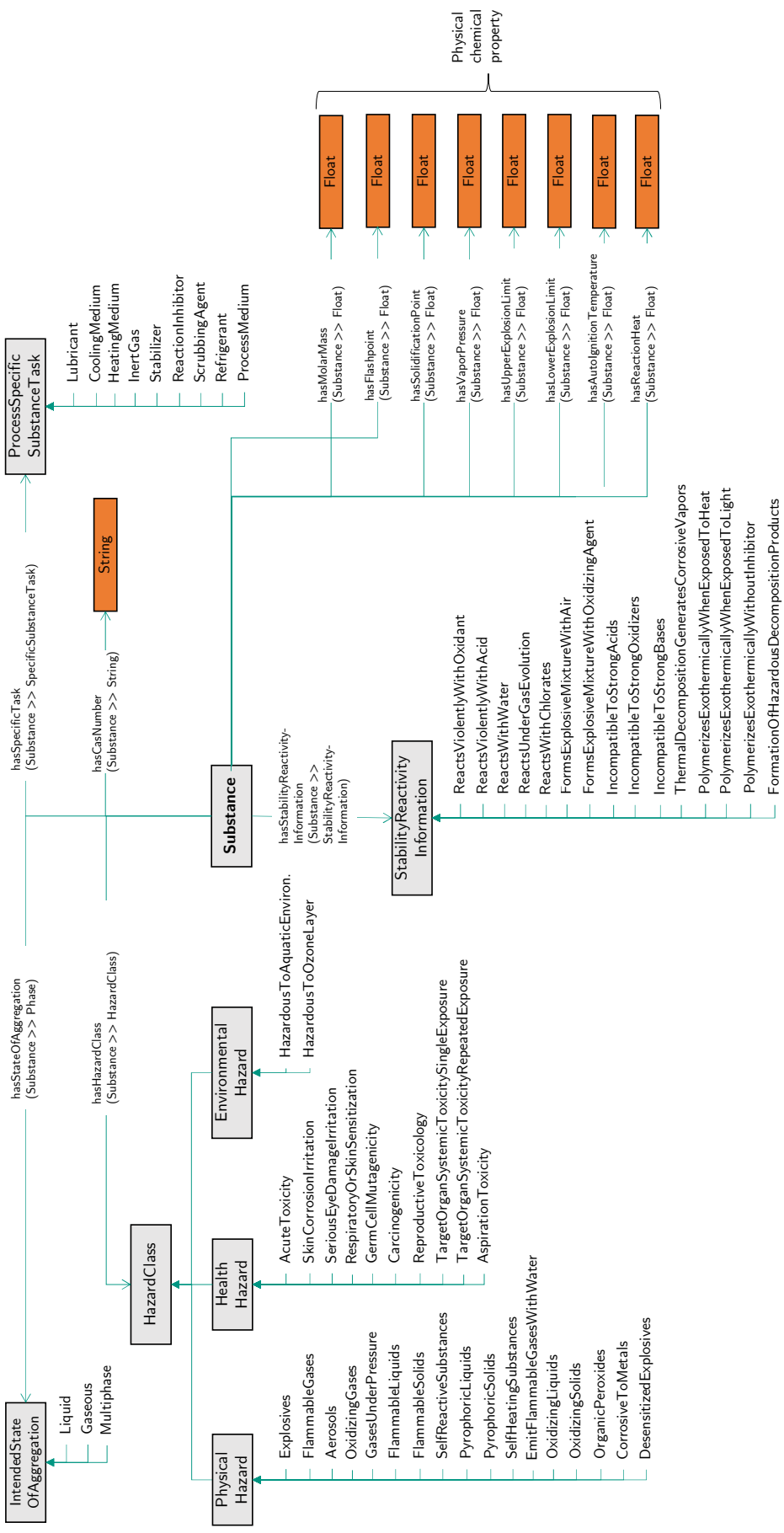


Figure 33: Illustration of the classes and relations of the *substance* concept.

5.5 Equipment entity

The plant representation must contain all information required to identify potential hazards, equipment failures, and operability problems. The hazard identification can be conducted unit or node-based, where a node consists of connected units. In this context, it must be examined how far a unit differs from a node, how specific equipment differs from a unit, and which intersections exist. Plants and their components are classified and subdivided hierarchically (cf. Table 11). Within DIN EN 61512-1 (IEC 61512), a physical model describes and classifies corporations' physical assets, including the hierarchical division of site, production area, process cell, unit, equipment module, and control module (cf. Figure 36). DIN EN ISO 14224 describes a taxonomic representation divided into plant/unit, section/system, equipment unit, subunit, component/maintainable item, and part. Within DIN EN 61512-1, a process cell is described as a logical grouping of facilities required for production. This description is similar to the term *plant system* from ISO 10303-231, which describes it as a combination of plant items required to produce products.

In the OntoCAPE approach, the concept *plant item* describes a plant's components (cf. Morbach et al., 2009). Plant items are divided into fixtures, equipment, transport channels, and instrumentation (cf. Figure 36). Figure 36 shows similarities within the different standards and approaches' abstraction levels. The horizontal lines are indicators for similar hierarchical levels. It becomes clear that terms are not used uniformly across standards or approaches and cannot be transferred directly. Accordingly, terms should be put into context using a P&ID section. In Figure 34, the previously described terms are assigned to elements of the P&ID section. DIN EN ISO 14224 uses the terms *section* or *system* for the plant's main components, while DIN EN 61512-1 uses the term *unit* for specific components.

DIN EN ISO 10628-1 refers to *subprocess* or *plant components*. HAZOP terminology usually refers to nodes representing selected sections of the process plant. The term *plant section* is used in this research because the terms within the standards differ, and it is common within the HAZOP domain. Within the DIN EN 61512, an *equipment module/entity* is described as equipment, including its functional logic. This definition corresponds to the term *technical item* from DIN EN ISO 10628, which describes functional units that describe the equipment in its entirety and not only partial aspects such as the physical enclosure. In a broader sense, the term *major process equipment* from ISO 10303-231 also represents functional units because different equipment components are considered. The term **equipment entity** is used in this thesis as a working term, including physical equipment and equipment control.

Based on the terminology used in OntoCAPE, the term **plant item** is adopted for individual components because it is generic and inclusive. For example, screws or bolts can also be included in the term *plant item* but cannot be subsumed under the *term equipment*. Besides, confusion with the term *equipment entity* can be excluded. The detail and abstraction level of the process plant representation has been successively refined in this research work.

Table 11: Standards and approaches addressing the classification of plants

Standard/Approach	Scope
DIN EN ISO 14224	Collection and exchange of reliability and maintenance data for equipment
DIN EN ISO 10628	Diagrams for the chemical and petrochemical industry
DIN EN 61512-1	Batch control - Part 1: Models and terminology
ISO 10303-231	Product data representation and exchange
OntoCAPE	Ontology for the domain process engineering

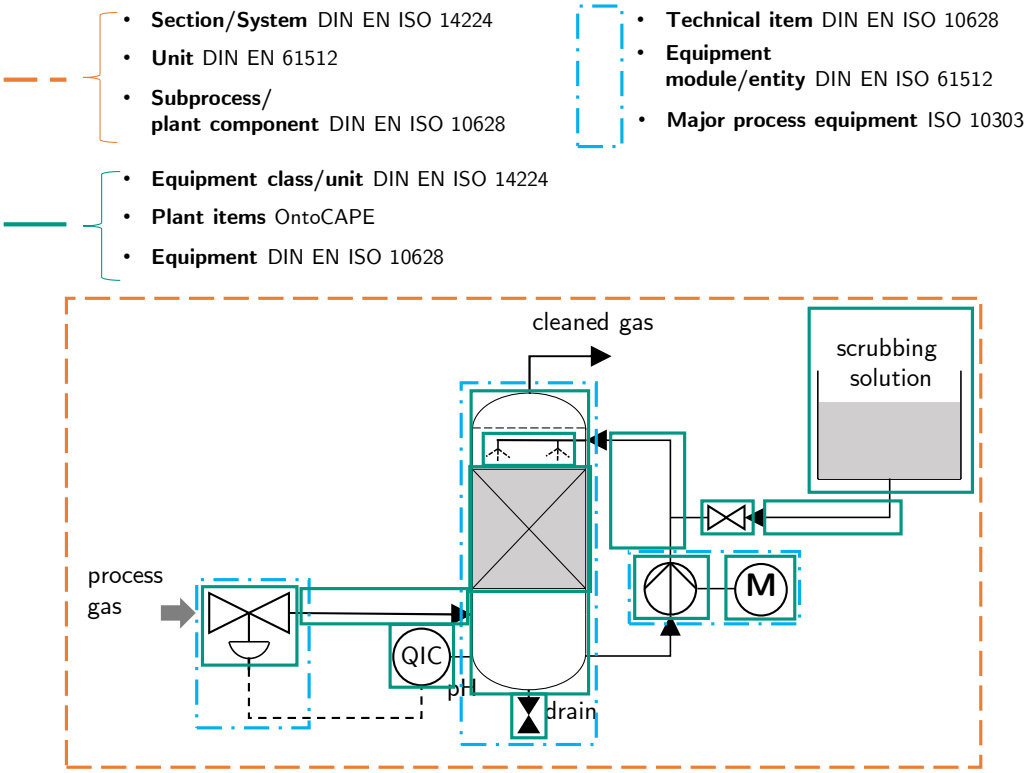


Figure 34: Classification of terms using a P&ID section

5.5.1 Development steps in terms of abstraction and level of detail

Initially, it was assumed that scenarios within the ontology could be sufficiently represented solely based on the name of the involved equipment entity (cf. Figure 37, I). The advantage of it is the simplicity of the modeling, allowing an automatic population of an ontology (cf. Single *et al.*, 2020c). This approach results in the static modeling of scenarios since all scenarios have to be represented for specific equipment entities combined with a specific substance. Nevertheless, scenarios do not depend exclusively on the type of equipment entity but on structural and functional aspects of it. This model conception is too superficial to be useful for reliably identifying scenarios. For example, whether a process vessel is equipped with a heating jacket is relevant. A heating jacket cannot be directly represented and assigned to the process vessel within this simplified model presentation since only the type of equipment entity is considered. Due to these inadequacies, this model conception was discarded.

In the course of advancements, the model conception was extended by introducing the subsystem concept to consider subsystems of equipment entities. Hence, differences between similar equipment entities can be represented (cf. Figure 37, II). Accordingly, all components and subsystems associated with an equipment entity are assigned to the category *subsystem*. For instance, a process vessel could be composed of the subsystems heating jacket, drain valve, and temperature sensor. A centrifugal pump could have the subsystems mechanical sealing, impeller, and a controller. This model conception was proposed in Single *et al.* (2019b). Such a model conception is shown in Figure 35 using the two equipment entities storage tank and chemical reactor. Both equipment entities involve drain valves, shut-off valves, filling level measurement, pressure relief valve, and cooling. They differ fundamentally in their function and function-specific details, such as tank blanketing or stirrer system, but have multiple subsystems in common. Nevertheless, labeling all equipment entity-specific details as subsystem leads to semantic inconsistencies, e.g., a reactor's cooling system can be called a subsystem. In contrast, a drain valve or vent pipe can hardly be called a subsystem.

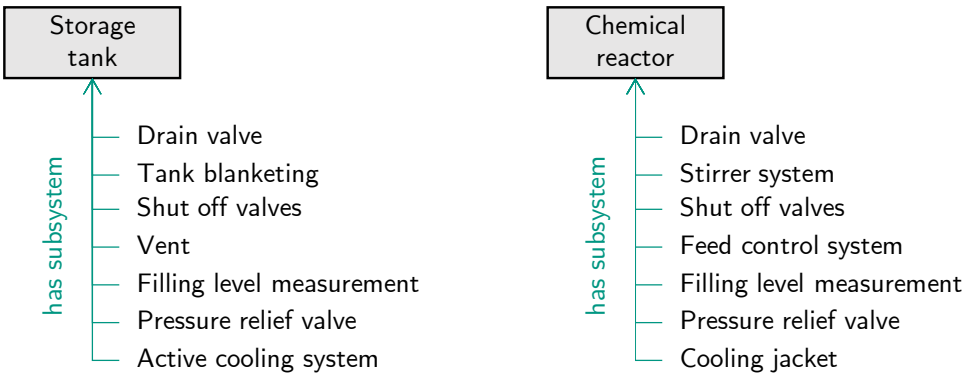


Figure 35: Assembling equipment entities based on subsystems

It may be necessary to subdivide a subsystem into further subsystems to capture specific details. A semantically correct class hierarchy construction is adversely affected by subsuming all plant items under the subsystem concept. Functional and structural differences are not apparent when all plant items belong to the same abstraction level of the subsystem concept. Comparing equipment-specific findings is more complex, and the transfer of findings to other equipment with common characteristics is hindered. Due to these disadvantages, this model conception was discarded in this work.

5.5.2 The selected, overall level of detail and abstraction and structural plant items

The exclusive distinction between equipment entities and subsystems was discarded, as this distinction is insufficient, and a finer level of detail was chosen. The idea of describing equipment through different function modules is described in DIN EN ISO 14224 and was adopted into this research. The strategy was to start from a coarser level of detail and increase the number of descriptive attributes (plant items). Hence, scenarios can be transferred to other equipment entities based on structural and functional similarities (cf. Figure 37, III). The special requirements within the HAZOP domain are considered; a too superficial level of detail regarding the equipment entities prevents specific concepts from being representable. Thus, specific scenarios cannot be identified. If the characterization of equipment entities is too detailed, the effort of creating a plant representation becomes too high, and the focus shifts from HAZOP analysis to plant modeling. Accordingly, the following requirements are considered for identifying an appropriate level of detail and abstraction:

- Scenario components should be transferable between different equipment entities based on structural and functional similarities.
- The intended function must be considered to describe functional aspects.
- It must be possible to represent connections between plant items between different equipment entities, e.g., between an upstream control valve and a downstream level transmitter.
- Since HAZOP scenarios also involve process control technology and instrumentation, these details must be considered in addition to the physical enclosure and fixtures.
- The level of detail must be chosen to identify all safety-relevant causes and consequences.

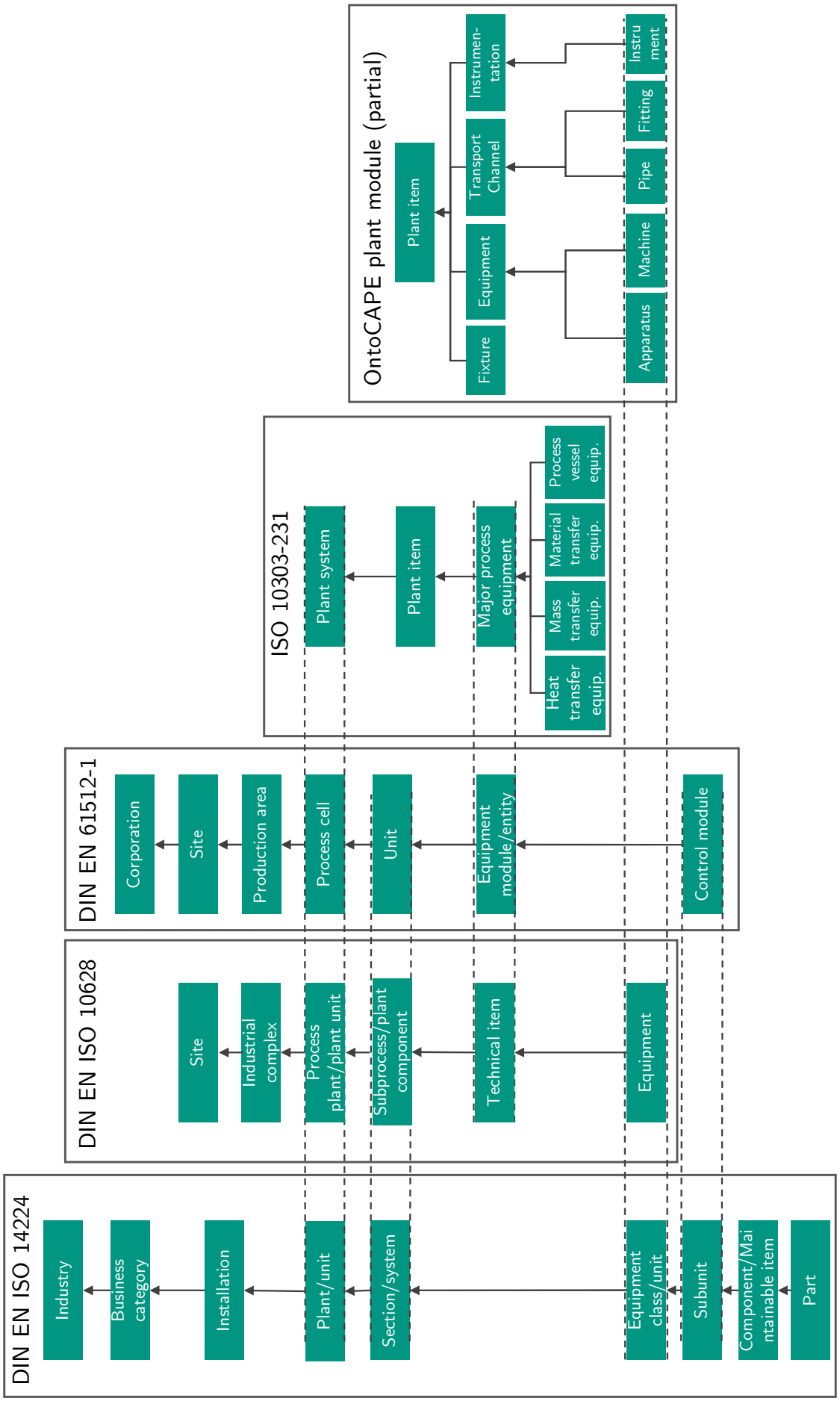


Figure 36: Classification of physical assets in different standards and approaches

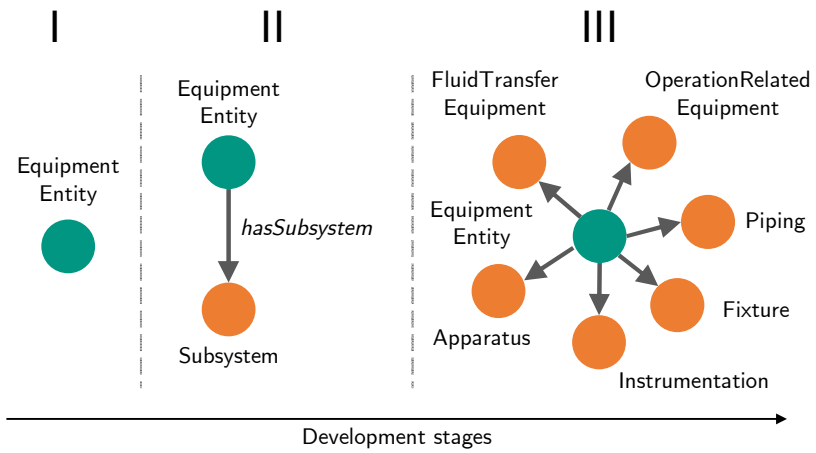


Figure 37: Development stages of the equipment entity representation

These considerations result in a higher-level hierarchy. A plant can be divided into plant sections, consisting of equipment entities composed of plant items. Within the HAZOP terminology, such plant sections are also called *nodes*. Plant items can, in turn, be divided into structural and functional plant items. Structural plant items are plant items that are of a passive or structural nature. They are often used to describe the enclosing shell or the plant items primarily made of steel. On the other hand, functional plant items have a functional character and can be controlled and operated. The selection of plant items and characteristics of equipment entities presented below is the result of more than three years of research and development. Different stages of development over chronological time are shown in Section 5.10. An overview of apparatuses is shown in Figure 39. Definitions, explanations, and examples of structural plant items are presented below.

Piping

The pressure equipment directive (PED) 2014/68/EU defines *piping* as components used for transporting fluids and connecting pressure systems, such as pipes, fittings, expansion joints, and hoses. This definition is adopted in this work. Pipes and in-line components like pipe fittings are subsumed under the concept **Piping** (cf. Figure 39). Furthermore, specializations, such as VentPipe, TubeCoil, TubeBundle, and BlanketingGasLine, are subsumed under **Pipe**.

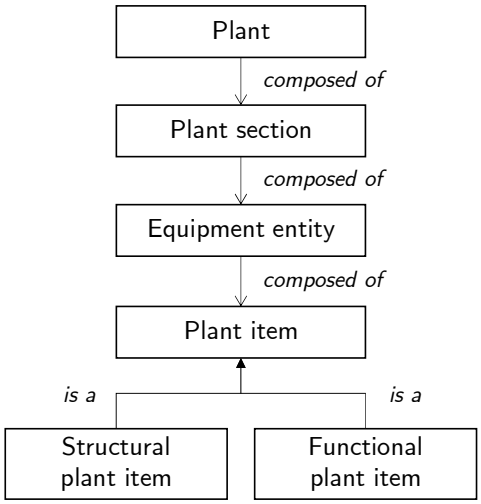


Figure 38: Selected hierarchical structure for representing plants

Apparatus

In ISO 10303-231, the term *process vessel equipment* describes enclosures containing fluids or solids under pressure or vacuum, e.g., pressure vessels, bins, and atmospheric tanks. The PED defines a vessel as a designed and built housing to contain fluids under pressure. In this definition, a storage tank would not be included. Within the CCPS (n.d.) glossary, an atmospheric storage tank is defined as a "storage tank designed to operate at any pressure between ambient pressure and 0.5 psig". The type of apparatus influences the type of appropriate safeguards. For instance, an atmospheric storage tank that does not fall under the PED (pressure < 0.5 barg) is not necessarily safeguarded by a *certified* safety valve; for the protection of a pressure vessel, a *certified* safety valve is usually required. Within the OntoCAPE approach, the general term *apparatus* is used for equipment that "mainly consists of non-moving parts and provides space for the material to be processed" (cf. Morbach et al., 2008). This general definition of the term *apparatus* is adopted and expanded within this research. Hence, an apparatus describes the enclosure of an equipment entity, such as:

- **AtmosphericStorageTank** (internal pressure < 0.5 barg),
- **PressureVessel** (e.g., reactor, separator, shell and tube heat exchanger),
- **OpenVessel** (e.g., cooling tower, external floating roof tank),
- **Casing** (e.g., pump, compressor),
- **Body** (e.g., valve).

Although casing and body are structurally and functionally similar, a distinction is made between them because of established terms. Within the ISO 13709, a casing is defined as all fixed and pressure-containing parts of a pump. In the ASME PTC 25, a *body* is defined as a containment or pressure-retaining component of a pressure relief device.

Fixture

Apparatuses can be equipped with fixtures. In the DIN EN 62714-2, a fixture is described as a device to limit an object's degree of freedom. Within the OntoCAPE approach, fixtures describe items attached to or installed in apparatuses (cf. Morbach et al., 2009). The fixture concept has been adopted to describe specific aspects of apparatuses within this work. Fixtures are not intended to stand alone but describe specific equipment parts, such as heating or cooling jackets, tube bundles, or packed beds. Figure 39 shows an overview of the considered fixtures. The list of fixtures is reusable and extendable. Additionally, fixtures are not intended to describe functional assemblies. Still, they represent a part of an assembly, e.g., the functional unit stirrer in a reactor consists of a stirrer, a drive, and a control. Only the stirrer is the fixture. Furthermore, they can model specific characteristics of apparatuses relevant to modeling specific causes and effects. For instance, the guideword 'high level' at an atmospheric storage tank could lead to overpressure. By adding the fixture "VentPipe", the inferred effect would be "Overflow" assuming that the ventpipe is permanently open, which is fundamentally different. A blocked vent pipe would, in turn, lead to an overpressure.

5.5.3 Assembling equipment entities from structural plant items

The basic components of equipment entities can be assembled based on the structural plant items. Table 12 shows examples of different heat exchangers consisting of the structural plant items *piping*, *fixture*, and *apparatus*. As shown in Table 12, not all structural plant items are always required to model equipment entities. Thus, generic equipment entities can be assembled based on the available structural plant items. If necessary, the proposed structural plant items can be extended to represent new or company-specific equipment entities.

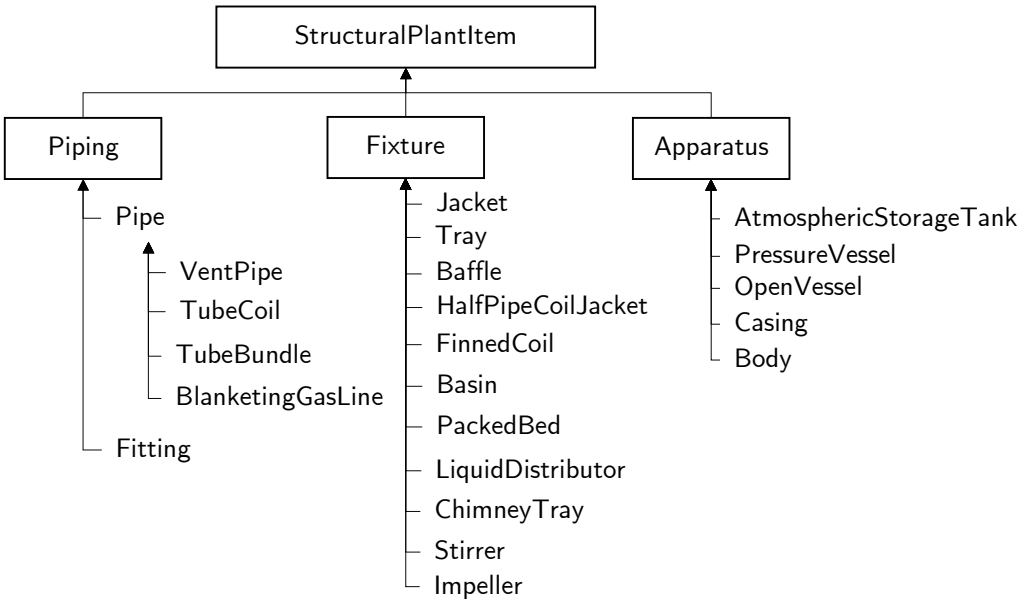


Figure 39: Taxonomy of the relevant structural plant items

Table 12: Assembling equipment entities from structural plant items

Structural plant item	**Object property**	**Equipment entity**		
		Shell and tube heat exchanger	Plate heat exchanger	Air cooled condenser
Apparatus	hasApparatus	PressureVessel	-	Casing
Piping	hasPiping	TubeBundle	-	-
Fixture	hasFixture	Baffle	PlatePackage	FinnedCoil

The examples in Table 12 are not complete models of equipment entities because, in addition to structural aspects, functional and control aspects should also be considered during modeling. Hence, structural plant items, such as material transfer equipment, subunit, or instrumentation, required for a complete representation of equipment entities are described in the following.

5.5.4 The selected level of detail and abstraction – Functional plant items

Functional plant items include plant items with a functional character, e.g., material transfer equipment or instrumentation. Also, autonomous subunits, such as cooling or lubrication systems, are assigned to this category. Usually, functional items can be controlled and operated.

Material transfer equipment

In ISO 10303-231, the term *material transfer equipment* is defined as process equipment for transferring process material from one location to another or from one thermodynamic state to another. Material transfer equipment is differentiated from mass transfer equipment where the driving force is the material concentration or a phase change (cf. ISO 10303-231). The term *material transfer equipment* is adopted from ISO 10303-231, which describes process equipment that enables material movement. It includes pumps, compressors, fans or blowers, conveyors for solids, jet ejectors, turbines, and expanders. A change in pressure usually accompanies the transport of fluids. Material transfer equipment is a commonly used type of equipment assigned to the functional plant item category (cf. Figure 40). This work focuses on pumps (centrifugal, positive

displacement) and compressors (screw, piston), but fans on air-cooled condensers are also considered. Fluid control equipment, such as control valves, are not considered material transfer equipment because they do not transfer material but control it.

Subunit

In ISO 14224, the term *subunit* is used to describe systems necessary for the function of equipment. The subunit concept has been adopted to model specific functional aspects of equipment entities, such as lubrication, burner, cooling, or heating. Additionally, components, such as bypass, non-return valve, or condensate separator, are subsumed under subunit. Subunits can model functional aspects of equipment entities, such as auxiliary and utility systems. Subunits are modeled in their entirety and are not divided further into their individual parts since the focus is on functional and not structural aspects. Subunits are essential for capturing functional characteristics, e.g., a lubrication system can alter causes regarding a compressor. The considered subunits are shown in Figure 40.

Instrumentation

Within ISO 10303-231, *instruments* are described as components that monitor or control a process plant's systems, such as control valves, sensors, gauges. This description corresponds to the definition in DIN EN 61511-1, where an instrument is defined as a device for performing an action in instrumented systems. Instrumentation items assume process control tasks and include items to measure and control processes. In this work, monitoring systems, sensors, controllers, actuators, and physical devices are subsumed under **Instrumentation** (cf. Figure 40). Instrumentation fittings, such as valves, are also subsumed under **Instrumentation** within OntoCAPE (cf. Wiesner *et al.*, 2008), which is adopted in this work. Furthermore, the category *monitoring system* includes acoustic signals, alarms, flashing lights, or similar components that indicate when thresholds have been exceeded or not reached and alert operators to act.

The term *sensor* subsumes all sensors such as level, temperature, pressure, flow, pH, or concentration sensors. The *controller* category includes all types of programmable logic controllers and task-specific controllers, such as speed controllers and assemblies, such as level, pressure, and flow indicator controller. The *actuator* group describes devices that require a control signal and convert energy into motion. Electric actuators include electric motors, solenoids, hydraulic actuators, and pneumatic actuators, which typically open and close control valves. Manual actuators are driven by muscle power and controlled by the operator, e.g., a shut-off valve with a handwheel.

The group of physical devices includes control valves with different actuators, shut-off valves with manual actuators typically operated by operating personnel, and restriction orifices that limit the mass flow. Furthermore, an orifice plate flow meter can be assigned to this category because it takes over measuring tasks while a controller can process the determined quantity. In addition to the direct instrumentation, equipment entities can be connected to adjacent instrumentation, e.g., a control valve connected to a level indicator controller of an adjacent storage tank.

5.5.5 Intended function

The intended function is a characteristic of the equipment entity. It describes the intended interaction between process, substance, and equipment entity. When differentiating between the intended functions, the process-specific meaning must be considered. For instance, heat transfer, condensation, and evaporation are differentiated, as different causes or consequences can arise due to the phase change.

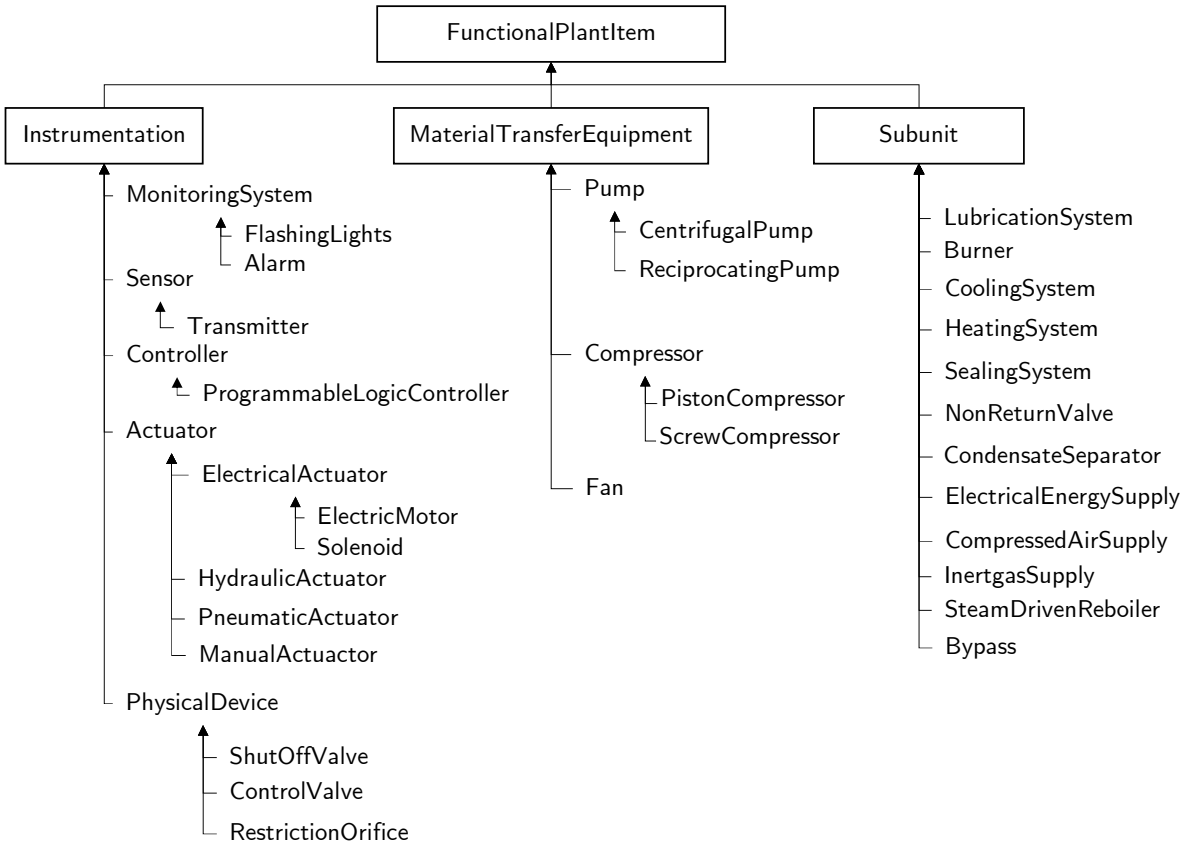


Figure 40: Taxonomy (class hierarchy) of the relevant functional plant items

Furthermore, the intended function of equipment entities may change when processes are considered transiently or different modes of operation are considered. Equipment entities can be similar in structure, and specific cause-effect relationships can only be modeled and identified based on the intended function. The intended functions are not exhaustive and can be added to the ontology if required. Further descriptions regarding the intended function can be annotated using *rdfs:comment*. The following intended functions are considered in this work (cf. Appendix Listing I-5):

- Evaporating, e.g., refrigerant evaporator,
- Condensing, e.g., condenser,
- HeatTransferring, e.g., shell and tube heat exchanger,
- Reacting, e.g., chemical reactor,
- Mixing, e.g., blender,
- Separating, e.g., settling tank,
- MaterialTransfer, DeliverConstantVolumeFlow, e.g., centrifugal pump, piston compressor,
- FlowControl, e.g., flow control valve,
- PressureControl, e.g., pressure control valve,
- TemperatureControl, e.g., cooling tower,
- Storing, Filling, Emptying, e.g., storage tank,
- Transporting, e.g., pipeline,
- Purifying, e.g., wet scrubber,
- Inerting, e.g., nitrogen blanketing,
- Unloading, e.g., tank truck,
- Stabilizing, e.g., stabilization (stripping) column.

5.5.6 Mode of operation

Scenario identification depends on the operating mode of the plant, as both design parameters and intended function may differ within the operating modes. Even in continuous processes, a distinction could be made between start-up, shutdown, and maintenance operations. It is possible to assign several modes of operation to an equipment entity, and there can be different scenarios for each mode of operation. Scenario components do not necessarily depend on the mode of operation, which can be expressed by assigning **ModeIndependent** to the class definition. The principle of *minimal ontological commitments* applies (cf. Section 3.8). The considered modes of operation are:

- NormalOperation,
- StartUpOperation,
- ShutDownOperation,
- Maintenance,
- ModeIndependent.

5.5.7 Control instance

A **ControlInstance** can be specified for each equipment entity to describe how the equipment is operated and controlled. Considering the control instance allows more precise identification of causes and underlying causes because control errors can be assigned uniquely, e.g., if a human error or a process control system failure are the causes. The following categories are subsumed under control instance:

- Operator,
- ProgrammableLogicController,
- OperatorAndProcessControlSystem,
- NotControlled.

5.5.8 Boundary conditions

During the ontology development, it became apparent that the reasoner could not distinguish all scenario components to that point, leading to incorrect conclusions (cf. Section 5.9.5). Therefore, the **BoundaryCondition** concept was introduced to differentiate scenario components (cf. Figure 31). The relevance of boundary conditions is described in more in Section 5.9.5. The boundary conditions can be assigned to individual equipment entities and not just globally for the entire process plant. For example, the air-cooled condenser of a refrigeration plant can be located outside, while the other parts of the plant are in the machine room. Furthermore, the boundary condition **LocatedOutside** determines whether external heat input from solar radiation can be considered plausible. Further boundary conditions can be added to the ontology if required to model specific scenarios. Additionally, subsequent boundary conditions may result from other boundary conditions. For instance, if the equipment entity is outside, it can be assumed that sufficient oxygen is available. The following boundary conditions are considered within this work:

- Equipment-related: **UpstreamProcessInvolved**, **LocatedOutside**, **FoundationCanBeAffected**, **ExternalFirePossible**,
- Substance-related: **IntroductionOfAir**, **IntroductionOfImpurities**, **IntroductionOfWater**, **SubstanceContainsStabilizer**,
- Personnel: **PersonnelPresentInTheNearField**, **SeveralPeoplePresentInTheNearField**.

The boundary conditions **SufficientOxygenAvailable**, **AmbientTemperatureCanDropBelowFreezingPoint**, **DirectIgnition**, and **DelayedIgnition,** are automatically set based on other boundary conditions, ambient, and substance information.

5.5.9 Ports

Equipment entities may have a different number of connections to adjacent equipment entities. For instance, a typical pump has one inlet and one outlet, while a typical heat exchanger has two inlets and two outlets. More complex equipment entities involve multiple inlets and outlets. The port concept was introduced to represent connections between equipment entities. During the specification of ports, a name or abbreviation, **Instrumentation**, e.g., Shut-off valve, and the **ConnectionType** (**Inlet**, **Outlet**) are assigned. The used object and data properties, such as hasName, hasConnectionType, and portEquippedWithInstrumentation are used for specifying ports (cf. Figure 41). For example, a port specification can look like: Port('p1', **InletValve**, **Inlet**). During scenario identification, the modeling of connections between equipment entities is particularly relevant for identifying specific scenarios. The use of ports to connect equipment entities is described in Section 5.7, Figure 44, and Figure 45.

5.5.10 Site information

Specific concepts require site information, divided into environmental and plant ambient information (cf. Table 13). Site information also includes environmental hazards, e.g., seismic activities, ambient temperatures, and traffic in the area surrounding the plant, e.g., forklifts or trucks. Typically, they provide site-specific information to determine the likelihood of causes or restrict specific causes (cf. Section 5.9.3). Ambient conditions apply equally to each equipment entity within the entire plant section under consideration. In contrast to boundary conditions, site information is set globally for the complete node under consideration (cf. Figure 49). Site information includes ambient plant information such as maximum and minimum ambient temperature and environmental dangers to consider environmental influences, such as storms and flooding (cf. Table 13).

Table 13: Site information

Environmental danger	Ambient plant information
DangerOfFlooding [ontology instance]	ForkLiftPresent [ontology instance]
DangerOfLandslide [ontology instance]	CranePresent [ontology instance]
DangerOfAvalanche [ontology instance]	RailTankWagonPresent [ontology instance]
DangerOfSeismicActivity [ontology instance]	AutonomousContainerTransport [ontology instance]
DangerOfHeavyRainfall [ontology instance]	TruckPresent [ontology instance]
DangerOfTsunami [ontology instance]	OtherVehiclePresent [ontology instance]
DangerOfTornado [ontology instance]	ResidentialAreaInTheNeighborhood [ontology instance]
DangerOfSevereStorm [ontology instance]	GatheringOfPeopleInSurroundingArea [ontology instance]
DangerOfSandstorm [ontology instance]	MinimumTemperatureInKelvin [float]
	MaximumTemperatureInKelvin [float]

5.5.11 Assembling complete equipment entities

Entire equipment entities are assembled based on the structural and functional plant items. An equipment entity can be composed of the following components and functional properties (cf. Figure 41): **Apparatus**, **Piping**, **Fixture**, **Instrumentation**, **MaterialTransferEquipment**, **Subunit**, **IntendedFunction**, **OperationMode**, and **ControlInstance**. For modeling control loops between instrumentation, even between different equipment entities, the relationship

formsControlLoopWith is used. Additionally, equipment entities can be provided with **Port**s to represent connections to other equipment entities. The maximum allowable working pressure and temperature, and a specific identifier, e.g., from a P&ID drawing, can be assigned to the equipment entity. The relationship isTransportable can be used to express whether the equipment entity is movable (e.g., tank truck) or fixed (e.g., a distillation column). Besides, equipment entities can be connected to adjacent plant items, e.g., a storage tank (equipment entity) be connected to a feed pump (material transfer equipment). It is not always necessary to use all attributes for describing equipment entities sufficiently. A concept map of all classes from which equipment entities are assembled is shown in Figure 41. The classes and relationships characterize the equipment entity through its components and functional properties. The relationships between the equipment entity and the concepts apparatus, piping, fixture, intended function, material transfer equipment, instrumentation, subunit, and control instance are assigned via object properties. Numerical values, booleans, or strings are assigned via data properties. In naming data properties, for semantic unambiguity reasons, the properties' physical unit is included in their designation, e.g., equipmentHasMaximumWorkingPressureInBarGauge. The object-oriented implementation of the equipment entities is done using the Python programming language. The Python class definition to create a port object is shown in Appendix Listing A-1. Appendix Listing A-2 shows the Python class definition for creating an equipment entity object.

5.6 Equipment entity modeling

Based on the considerations from the previous chapter, an equipment entity model library is developed using the presented structural and functional plant items. This model library is reusable and expandable since new equipment entities can be assembled based on the available components. The equipment entities presented are intended to explain the modeling approach, the use of the plant items, and the underlying theoretical framework. The formal class definitions of all current models within the equipment entity model library are shown in Appendix Listing D-1, Appendix Listing D-2, Appendix Listing D-3, Appendix Listing D-4. In the appendix, the equipment entities: plate heat exchanger, pressure receiver, screw compressor, piston compressor, tank truck, reactor, reboiler, inert gas blanketing, connection pipe, manual valve, throttling valve, fin and tube evaporator, cooling tower, and crude stabilization column are presented. The following site information is not considered since it is not equipment-specific but is defined globally for the entire plant section (cf. Section 5.7). In the following, the procedure for modeling equipment entities is illustrated through examples.

Storage tank

Storage tanks occur in different forms within the chemical industry and can be represented using the available plant items. Storage tanks can be modeled in different designs and may have the intended functions of **Filling**, **Storing**, **Emptying**. Low-pressure storage tanks are modeled with the apparatus **AtmosphericStorageTank**. An external floating roof tank is modeled using an **OpenVessel** as the apparatus (cf. Figure 46). In the case of a pressurized storage tank, the apparatus is modeled by a **PressureVessel**. Within the first example (cf. Figure 42), the fictitious tank is equipped with a **BlanketingGasLine** (piping) to intertize the tank as well as with a **TubeCoil** (piping), which is supplied by a **CoolingSystem** (subunit) to cool the tank. The tank is equipped with a **LevelIndicatorController** (instrumentation), and there is a connection (formsControlLoopWith) to an adjacent control valve, which is not part of the storage tank. Optional inlet and outlet shut-off valves (**ShutOffValve**, instrumentation) can be considered via the ports (cf. Section 5.5.9). Unique tags, such as "ST01-VA1" or "ST01-VA2", can be used to distinguish between valves of identical design. Listing 1 shows the formal class definition of the cooled storage tank.

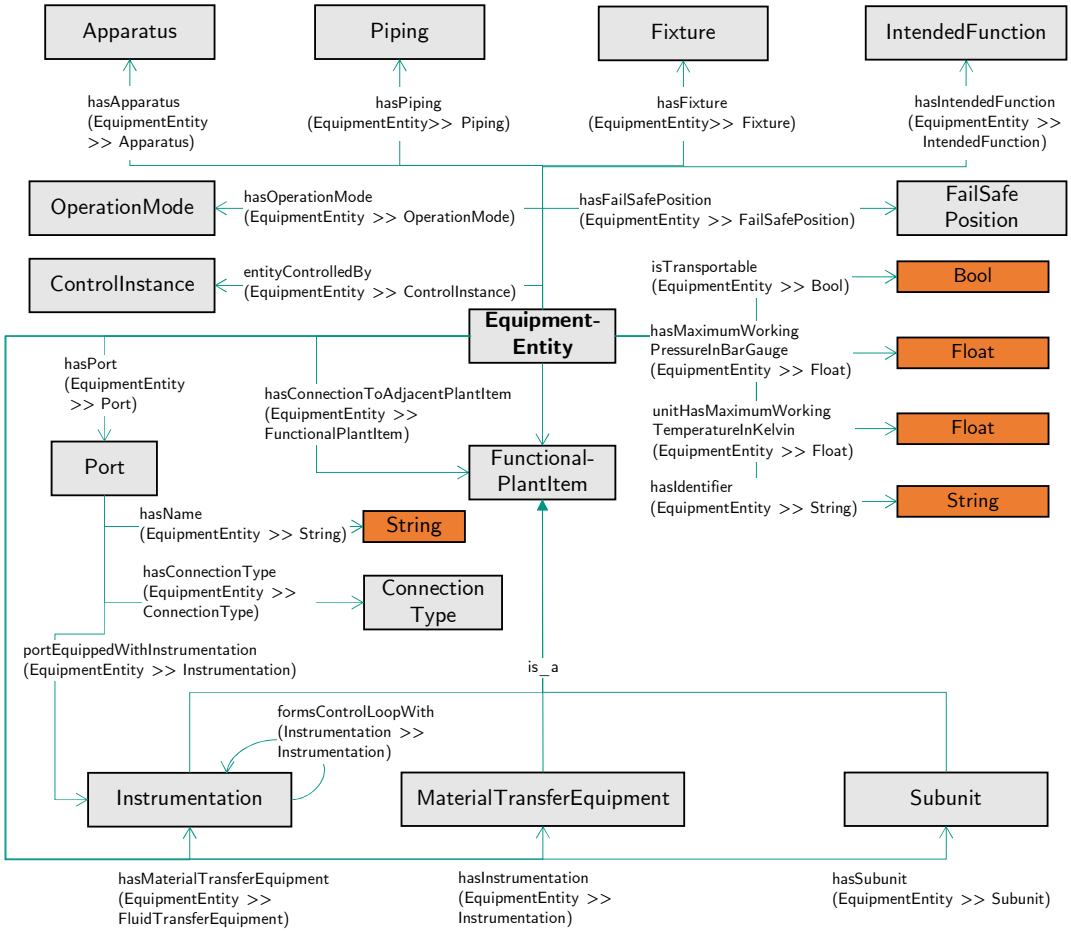


Figure 41: Illustration of the classes and relations of the *equipment entity* concept.

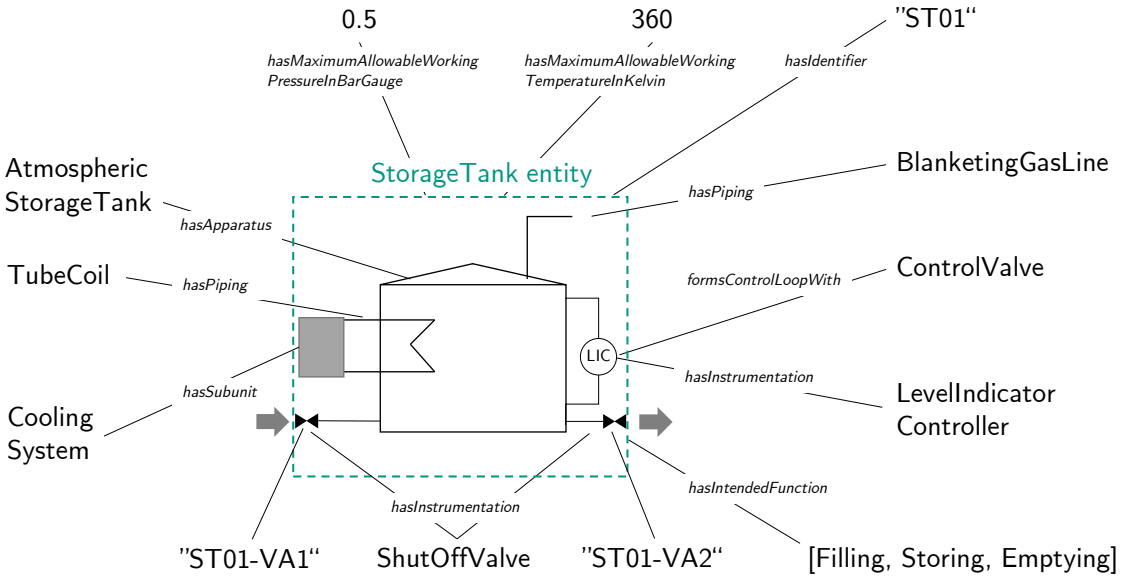


Figure 42: Assembling the equipment entity: cooled storage tank

An external floating roof tank is used as a second example since it has structural differences from the first tank (cf. Figure 46). It is equipped with the apparatus **OpenVessel**, the subunit **SealingSystem**, and the fixtures **FloatingRoof**, and **GuidePol**. Further details have been deliberately omitted to illustrate significant differences. These templates of storage tanks can be adapted to match the characteristics of company or process specifications.

```
def cooled_storage_tank(identifier, boundary_condition, control_instance, transportable, operating_modes,
                        max_pressure, max_temperature):
    equipment = equipment_onto.StorageTankEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, boundary_condition, transportable,
                                         operating_modes, max_pressure, max_temperature)
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment_entity.add_subunit(equipment_onto.CoolingSystem)
    equipment_entity.set_apparatus(equipment_onto.AtmosphericStorageTank)
    equipment_entity.add_piping(equipment_onto.BlanketingGasline)
    equipment_entity.add_piping(equipment_onto.TubeCoil)
    equipment_entity.add_instrumentation((equipment_onto.LevelIndicatorController, None))
    equipment_entity.add_connected_plant_item(equipment_onto.FlowControlValve)
    equipment_entity.set_intended_function(process_onto.Storing)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity
```

Listing 1: Class definition function of a cooled storage tank

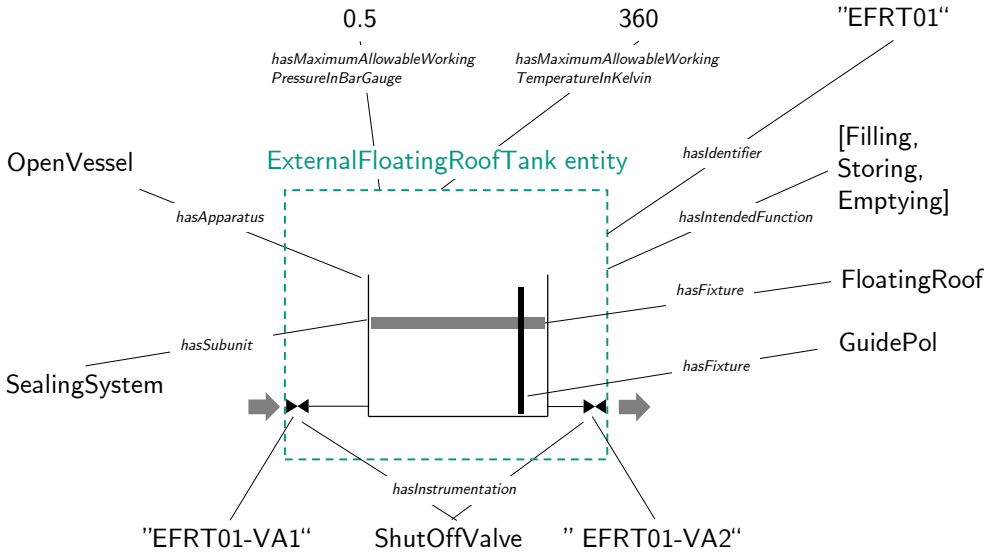


Figure 43: Assembling the equipment entity: external floating roof tank

Pressure equipment

Equipment entities such as pressure receivers, gas-liquid separators, or chemical reactors can be derived from a pressure vessel. For example, a wet scrubber entity is modeled in Figure 44. It has the intended function of **Purifying**, consists of a **PressureVessel** (apparatus), the fixtures **PackedBed** and **LiquidDistributor**, and it is equipped with a **QualityIndicatorController** (instrumentation). Since it has five ports connected to adjacent equipment entities, each port is labeled with an identifier, whether it is an **Inlet** or **Outlet**. Additionally, a port can be provided with a **ShutOffValve** or other instrumentation (cf. Figure 44). The mechanical integrity of the enclosure is described via the data properties:

- hasMaximumAllowableWorkingPressureInBarGauge “16” and,
- hasMaximumAllowableWorkingTemperatureInKelvin “360”.

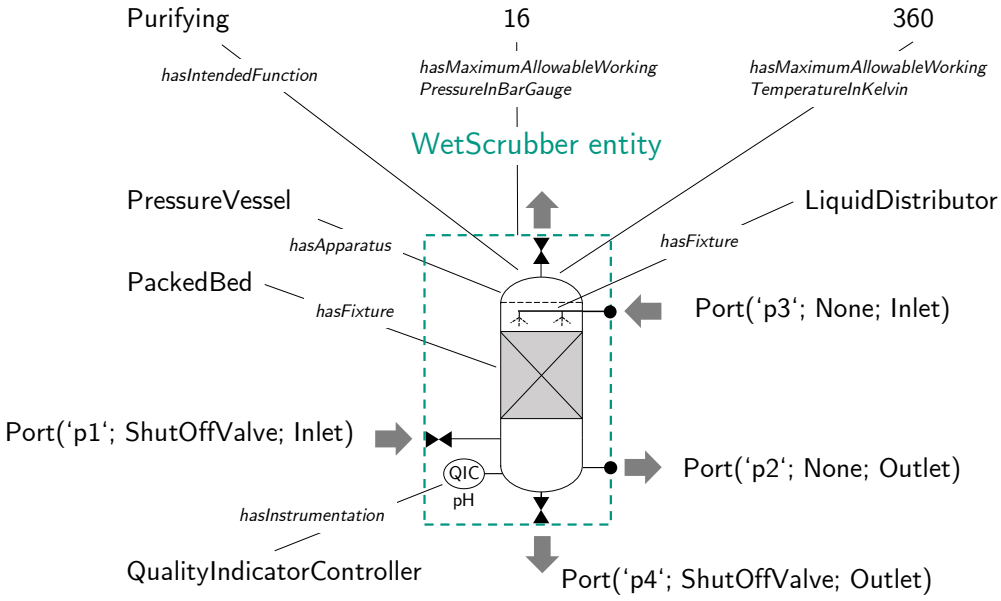


Figure 44: Assembling the equipment entity: *wet scrubber*

Heat exchanger

A shell and tube heat exchanger can be modeled using the described plant items and additional characteristics (cf. Figure 45). It has the intended function of **HeatTransferring** and consists of the fixture **Baffle**. The outer shell of the heat exchanger is modeled as a **PressureVessel** (apparatus). Since there are four ports at the shell and tube heat exchanger ports are used to label them and specify if they are an **Inlet** or **Outlet** (cf. Figure 45). Additionally, the heat exchanger is provided with the data properties to describes its mechanical integrity:

- hasMaximumAllowableWorkingPressureInBarGauge "28" and,
- hasMaximumAllowableWorkingTemperatureInKelvin "500"

Other heat exchangers can be modeled based on the available plant items; for instance, a plate heat exchanger is described in Appendix D.

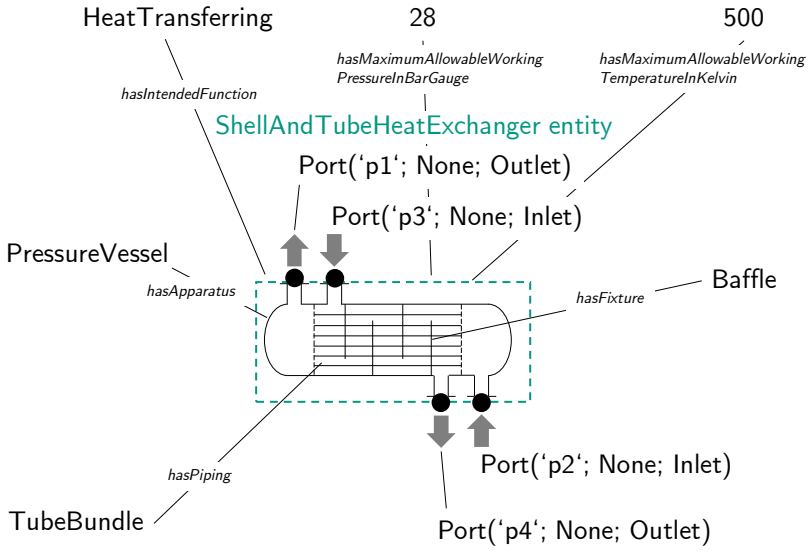


Figure 45: Assembling the equipment entity: *shell and tube heat exchanger*

Air-cooled condenser

Utility systems, such as refrigeration plants, are widely used within the chemical process industry. Refrigeration systems contain a condenser. An air-cooled condenser has the intended function of **Condensing** and can be composed of the apparatus **Casing** and the fixture **FinnedCoil**. It is also equipped with the material transfer equipment **Fan** that is driven by an **ElectricMotor** (instrumentation), which has an **ElectricalEnergySupply** (subunit) and is controlled by a **SpeedController** (instrumentation). The inlet and outlet **ShutOffValve** are modeled as a part of the condenser. The mechanical integrity of the casing is described via the data properties:

- hasMaximumAllowableWorkingPressureInBarGauge "15" and,
- hasMaximumAllowableWorkingTemperatureInKelvin "393".

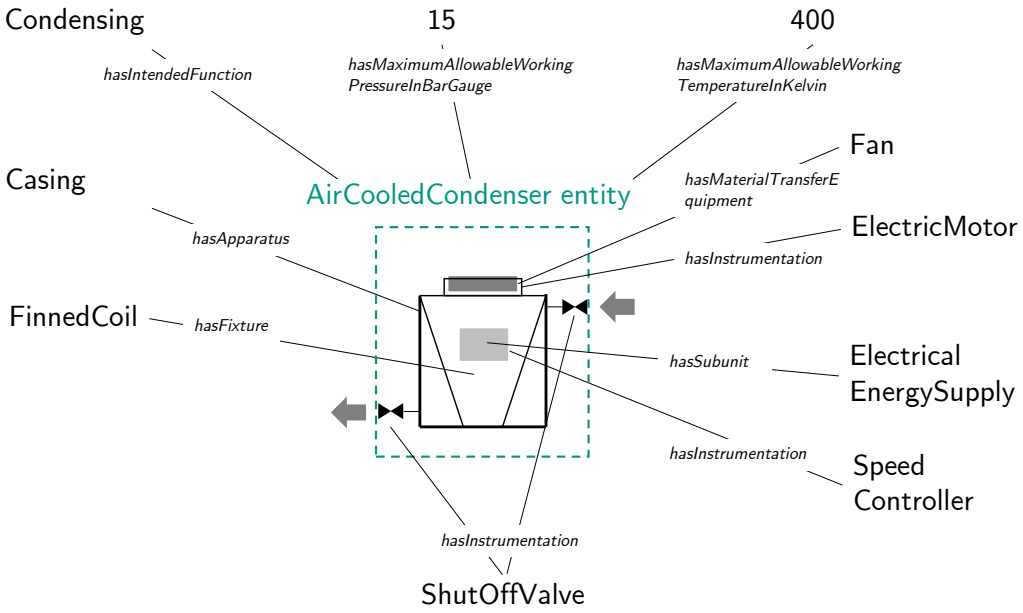


Figure 46: Assembling the equipment entity: air-cooled condenser

Material transfer equipment

A centrifugal pump is a widespread component and a typical representative of the class material transfer equipment. Depending on the complexity and the specific design, the number of plant items required to model a centrifugal pump varies. The schematic centrifugal pump entity is shown in Figure 47. It is composed of the apparatus **Casing** and has the intended function **MaterialTransfer**. The operating principle and the pump type are uniquely defined as a **CentrifugalPump** (material transfer equipment). Besides, it consists of the **Impeller** (fixture) and the **SealingSystem** (subunit). The **ElectricMotor** (instrumentation) requires an **ElectricalEnergySupply**, modeled as a Subunit and controlled by a **Controller** (instrumentation). The inlet shut-off valve and the pump's non-return valve are modeled as parts of the pump (cf. Figure 47). The level of abstraction and detail selected is essential to identify operability issues. The mechanical integrity of the casing is described via the data properties:

- hasMaximumAllowableWorkingPressureInBarGauge "15" and,
- hasMaximumAllowableWorkingTemperatureInKelvin "393".

Other material transfer entities, such as compressors, can be modeled using the available plant items (cf. Appendix D).

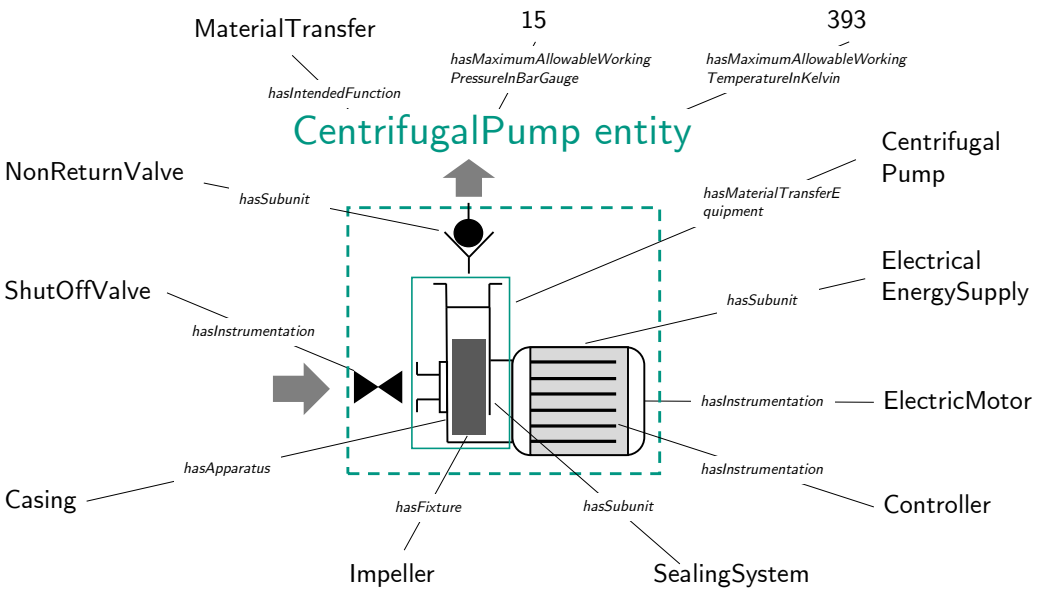


Figure 47: Assembling the equipment entity: centrifugal pump

Valve units

Valves involving an actuator or a control system, such as control valves or thermal expansion valves, are modeled as independent equipment entities. Simple shut-off valves can be modeled as part of a larger equipment entity described previously. Control valves have the intended function **FlowControl** and can be composed of the apparatus **Body**, the instrumentation **PneumaticActuator** and **Controller**, and the subunit **CompressedAirSupply** to supply the actuator with pneumatic operating power. In addition to the characterizing attributes, the specific type of valve must be specified. Hence, a unique description of the control valve entity, the descriptive attribute **ControlValve** (instrumentation), is also required (cf. Figure 48). Additionally, scenarios may depend on the control valve's fail-safe position, which is considered via the object property hasFailSafePosition. Valves can have a bypass, which is modeled by the subunit **Bypass**.

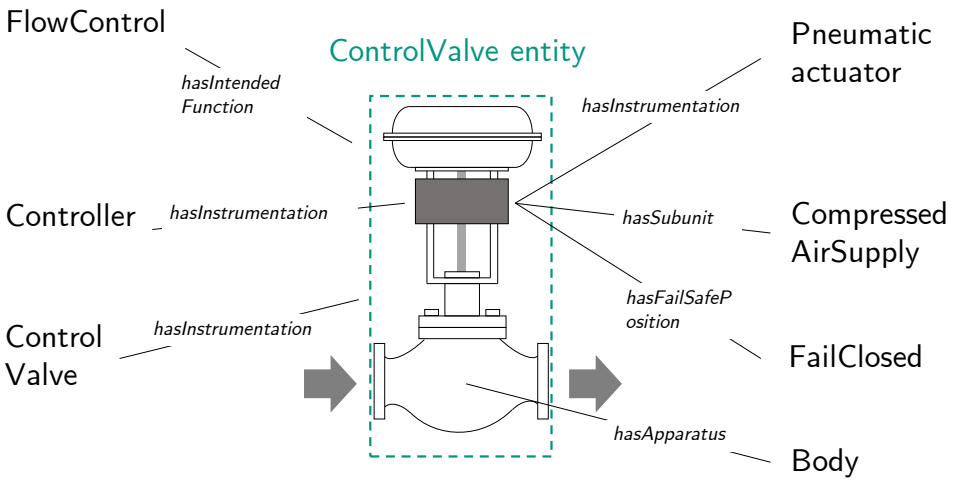


Figure 48: Assembling the equipment entity: control valve

5.7 System modeling

Process plants can be composed of multiple equipment entities connected via interfaces. Together with the substance model, boundary conditions, and site information, this results in the system model of the entire process plant. Accordingly, equipment entities can be considered separately and as a combination of connected equipment entities. For the representation of adjacent equipment entities, process plants are represented in signed directed graphs (SDG) (cf. Section 3.4). SDGs are suitable because directed edges represent the intended flow or material direction between equipment entities. The nodes can be composed of complex objects, e.g., OWL ontology instances. SDGs are composed using the Python package NetworkX (cf. Hagberg et al., 2008). Each equipment entity can be represented as a node (not to be confused with HAZOP nodes), while objects containing the equipment entity (cf. Section 5.5), substance (cf. Section 5.4), and site information are assigned to each node. Thus, every SDG node contains an object-oriented description of the equipment entity, the process, the substances involved, a description of boundary conditions, and site information. Figure 49 shows an example of information contained in the nodes. The plant section consists of a control valve (node 1) and a wet scrubber (node 2). The intention of the scrubber is to remove sulfur dioxide (SO_2) from flue gas using seawater. Both nodes are connected via ports, while the wet scrubber has additional ports (cf. Section 5.5.9). This example is intended to describe the objects and the information they contain in more detail and provide information on how process plants can be modeled to be analyzed with the help of the developed knowledge-based system.

The equipment entity, ports, function description, substances, and boundary conditions are defined explicitly within the graph. Substances are assigned equipment entity-specific because different substances may be present in different plant parts. For example, the flue gas is present in the control valve, while flue gas and seawater are present in the wet scrubber. The same applies to boundary conditions, e.g., if the wet scrubber is outside while the control valve is inside a building. As described in Section 5.5.9, the equipment entities are characterized based on the structural and functional plant items shown in the schematic equipment entity objects in Figure 49. There is an action line between the control valve and the **QualityIndicatorController,** which is part of the wet scrubber, represented by the formsControlLoopWith relationship (cf. Figure 41). Since the flue gas is not specified, it is assumed that the safety-relevant substance is sulfur dioxide. Accordingly, according to the GHS system, the hazard classes are **AcuteToxicity**, **SkinCorrosionIrritation**, and **Irritation**. While only the flue gas, including SO_2, is present in gaseous form in the control valve, both the flue gas and seawater (liquid) are present inside the second plant section of the wet scrubber unit. The information that SO_2 can react with water and chlorates (stability/reactivity: **ReactsWithWater**, **ReactsWithChlorates**, Figure 41) is contained within the substance entity. The fact that information is missing or incomplete and that conservative assumptions have to be made is expected in the daily practice of HAZOP practitioners. To-do lists of incomplete information are created, which are resolved for the next meeting to use the HAZOP participants' time efficiently.

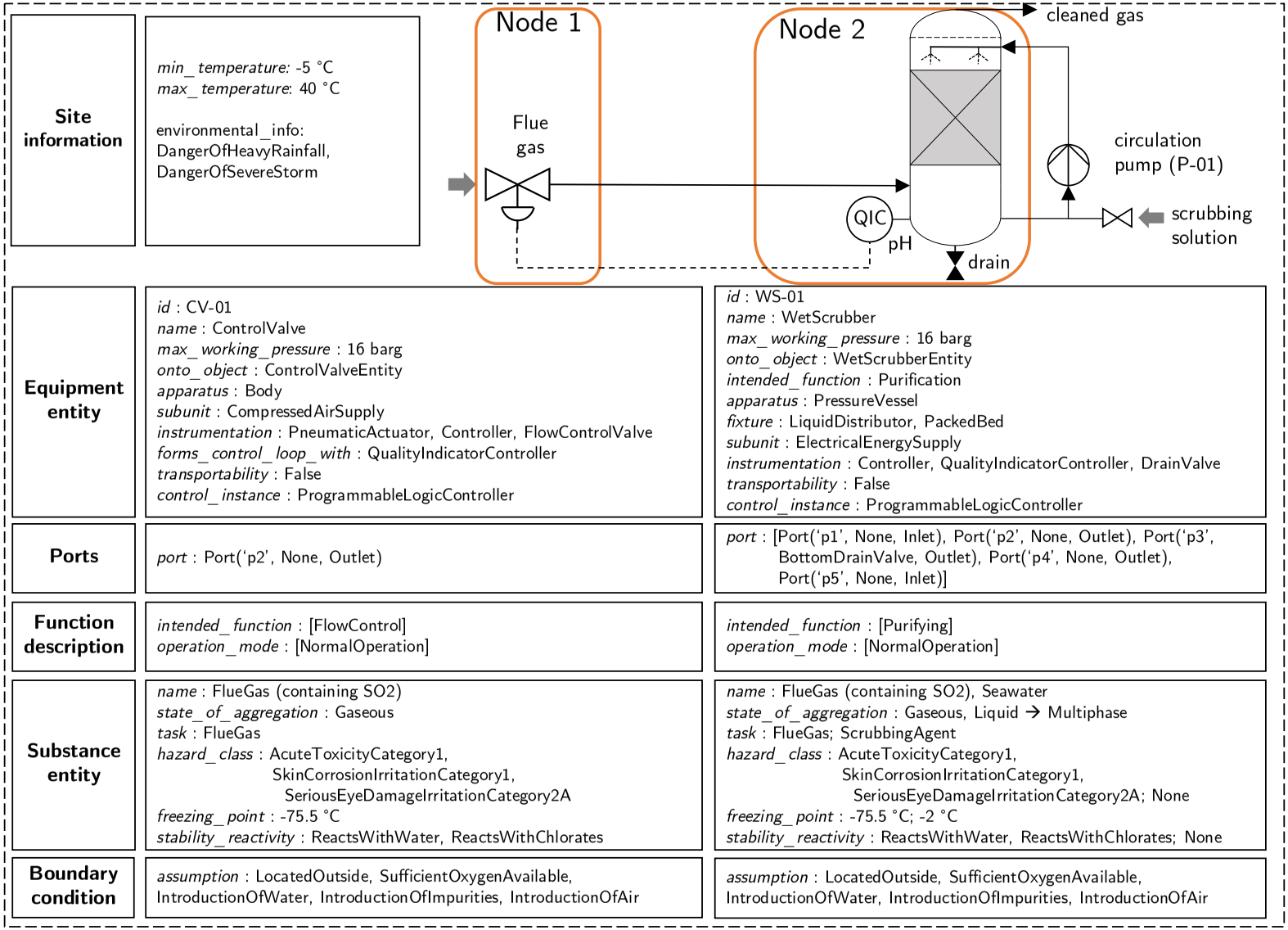
With the help of the definition of the intended function and the operating modes of all involved equipment entities, the function of the entire node under consideration is described. In addition, the boundary conditions define that the plant section is located outside (**LocatedOutside → SufficientOxygenAvailable**), and that both water, impurities, and air can be introduced into the plant section (**IntroductionOfWater**, **IntroductionOfImpurities**, **IntroductionOfAir**). Modeling the system in its entirety, considering adjacent equipment entities, allows deviations, effects, and consequences to be propagated through equipment parts and causes to be identified outside the specific equipment entities (cf. Section 5.14). This view enables systemic hazard identification.

5.8 Conceptualization of complementary semantic relationships

The overall structure of the developed ontology consists of the core HAZOP concepts (cf. Section 5.2.3). The semantic relationships between the concepts equipment entity, substance, boundary condition, and site information were presented in the last sections. Additionally, the ontology includes a plant item taxonomy, characterization of substances, and risk matrix. The overall structure of the ontology is shown in Figure 50. The generic concepts and their semantic relationships form a conceptual ontological framework for modeling domain-specific concepts for describing scenarios. The developed ontology can be classified as an application ontology with the generic concepts of a domain ontology (cf. Section 3.8).

The relationships described in the previous sections within the ontology modules *CoreHazop-Concept*, *Equipment,* and *Substance* are shown with solid lines. A modularization improves the maintainability, reusability and reduces the overall complexity. The modularization was done based on the different subject-matter knowledge of the modules. For instance, the *Equipment* module covers plant items, while the *Substance* module characterizes substances and their properties. Modularization allows individual modules to be used independently of others. For example, the *Equipment* and *Substances* module can be used for other software components and assignments. The risk matrix contained in the *RiskAssessment* module is independent of other modules and can be reused or adapted due to industry-specific guidelines. The *CoreHazopConcept* module contains all information to dynamically assemble scenarios from scenario components and has dependencies on all other modules. The relationships between modules are shown with dotted lines in Figure 50 and are described in more detail in the following. When identifying and establishing relationships between modules, the following aspects should be recognized:

- Relationships between scenario components and equipment entities must be represented, including structural and functional properties.
- Relationships to the substances involved must access the substance properties needed to model scenario components.
- Relationships to site information must consider influential plant environment conditions within the scenario components.
- Relationships between scenario components and boundary conditions ensure their plausibility.
- The likelihood of causes, the severity of consequences, and relationships to other concepts must be determined to identify safeguards.



Site information	*min_temperature*: -5 °C *max_temperature*: 40 °C environmental_info: DangerOfHeavyRainfall, DangerOfSevereStorm	
Equipment entity	*id* : CV-01 *name* : ControlValve *max_working_pressure* : 16 barg *onto_object* : ControlValveEntity *apparatus* : Body *subunit* : CompressedAirSupply *instrumentation* : PneumaticActuator, Controller, FlowControlValve *forms_control_loop_with* : QualityIndicatorController *transportability* : False *control_instance* : ProgrammableLogicController	*id* : WS-01 *name* : WetScrubber *max_working_pressure* : 16 barg *onto_object* : WetScrubberEntity *intended_function* : Purification *apparatus* : PressureVessel *fixture* : LiquidDistributor, PackedBed *subunit* : ElectricalEnergySupply *instrumentation* : Controller, QualityIndicatorController, DrainValve *transportability* : False *control_instance* : ProgrammableLogicController
Ports	*port* : Port('p2', None, Outlet)	*port* : [Port('p1', None, Inlet), Port('p2', None, Outlet), Port('p3', BottomDrainValve, Outlet), Port('p4', None, Outlet), Port('p5', None, Inlet)]
Function description	*intended_function* : [FlowControl] *operation_mode* : [NormalOperation]	*intended_function* : [Purifying] *operation_mode* : [NormalOperation]
Substance entity	*name* : FlueGas (containing SO2) *state_of_aggregation* : Gaseous *task* : FlueGas *hazard_class* : AcuteToxicityCategory1, SkinCorrosionIrritationCategory1, SeriousEyeDamageIrritationCategory2A *freezing_point* : -75.5 °C *stability_reactivity* : ReactsWithWater, ReactsWithChlorates	*name* : FlueGas (containing SO2), Seawater *state_of_aggregation* : Gaseous, Liquid → Multiphase *task* : FlueGas; ScrubbingAgent *hazard_class* : AcuteToxicityCategory1, SkinCorrosionIrritationCategory1, SeriousEyeDamageIrritationCategory2A; None *freezing_point* : -75.5 °C; -2 °C *stability_reactivity* : ReactsWithWater, ReactsWithChlorates; None
Boundary condition	*assumption* : LocatedOutside, SufficientOxygenAvailable, IntroductionOfWater, IntroductionOfImpurities, IntroductionOfAir	*assumption* : LocatedOutside, SufficientOxygenAvailable, IntroductionOfWater, IntroductionOfImpurities, IntroductionOfAir

Figure 49: Representation of an example plant section

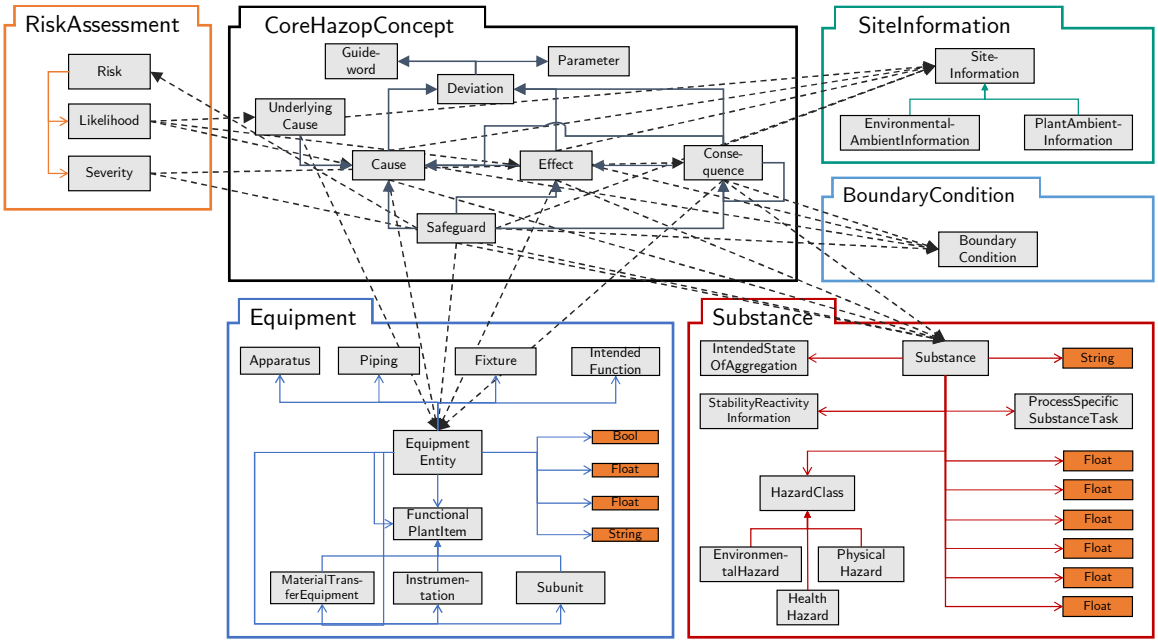


Figure 50: Overall ontological structure

5.8.1 Embedded relationships

Relationships between core concepts, equipment entities, and substances can be modeled differently. On the one hand, relationships between all plant parts, substance attributes, and scenario components can be explicitly modeled. On the other hand, plant parts and substance attributes can be directly assigned to an equipment entity using embedded relationships. Figure 51 shows the comparison between explicit and embedded relationships. At the beginning of the ontology development process, embedded relationships were not used, but explicit relationships were used. Using explicit relationships between each scenario component (e.g., cause and effect) and all structural and functional plant items, and additionally, all substance properties lead to boilerplate code within the formal class definitions. Besides, explicit relationships cause maintenance and scalability issues since each plant item or substance information must be related to each scenario component. Embedded restrictions simplify class definitions since there is no direct relationship between the scenario component, and each involved plant item must be considered. All plant items of the equipment entity are already implicitly considered. More generally, multiple definitions can be avoided by defining relations at a higher level. Thus, embedded restrictions have been used within later developed ontologies.

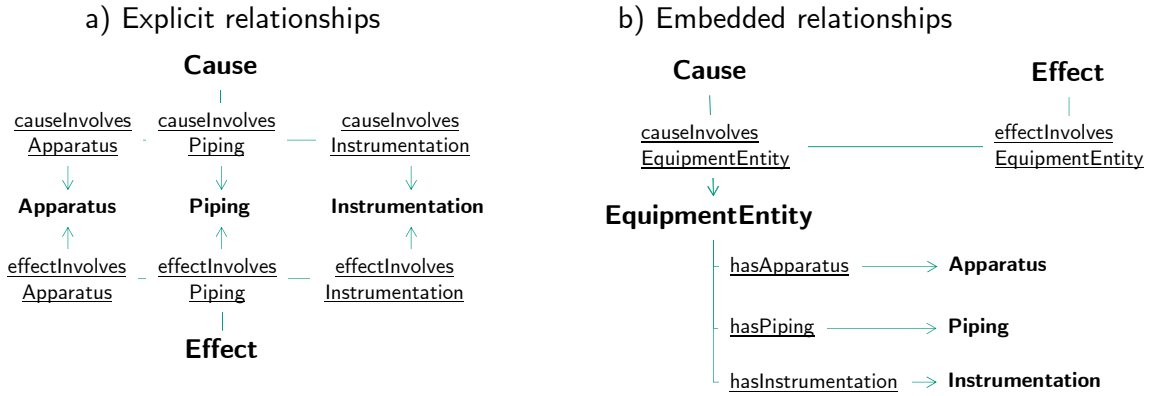


Figure 51: Explicit and embedded relationships

5.8.2 Semantic relationships of causes and underlying causes

In addition to the core semantic relationships described in Section 5.2.3, additional relationships are required to allow meaningful statements about possible underlying causes to be made. It is insufficient to model underlying causes only dependent on the causes because it is a too broad generalization. Hence, there are dependencies to the concepts **BoundaryCondition** and **SiteInformation** (cf. Section 5.2.3). For example, in Listing 2, the underlying cause "SolarRadiation" requires the boundary condition **LocatedOutside**, involves a **StorageTankEntity**, and is the underlying cause of the cause "ThermalOutbreathing".

Causes are associated with substances, equipment entities, deviations, boundary conditions, and site information (cf. Figure 52). Sub-concepts, such as **StateOfAggregation** or **Fixture**, are implicitly considered via embedded relationships. This generalization enables the transfer of causes from equipment entities to other equipment entities that share structural or functional plant items or involve the same substance properties without specifying the equipment entity. For instance, the cause "InternalLeakage", shown in Listing 3, requires the fixtures **TubeBundle** or **HeatingJacket** to characterize the involved equipment entity. Hence, no specific equipment (e.g., **TubeAndShellHeatExchanger**) is required in this cause; the structural plant items of the equipment entity are sufficient for describing the contribution of the equipment to the cause. By generalization, causes can be transferred to equipment entities that share structural or functional items. Besides, redundancies in the class definitions can be avoided, and the ontology structure is simplified. At this point, it should be noted that the relationships shown in Figure 52, such as causeRequiresBoundaryCondition or underlyingCauseRequiresSiteInformation, are optional to model-specific classes unambiguously and not mandatory for every subclass of **Cause** or **UnderlyingCause** contained in the ontology. For instance, the class definitions in Listing 4 do not contain any boundary conditions or site information.

```
class SolarRadiation(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     underlyingcauseInvolvesEquipmentEntity.some(StorageTankEntity) &
                     underlyingcauseRequiresBoundaryCondition.some(LocatedOutside) &
                     isUnderlyingcauseOfCause.some(ThermalOutbreathing)]
```

Listing 2: Class definition of "SolarRadiation"

```
class InternalLeakage(Cause):
    equivalent_to = [UnderlyingCause &
                     isCauseOfDeviation.some(ElsewhereFlow | OtherThanComposition) &
                     causeInvolvesEquipmentEntity.some(hasFixture.some(Jacket | HalfPipeCoilJacket | Platepackage) |
                                                       hasPiping.some(TubeCoil | TubeBundle))]
```

Listing 3: Class definition of "InternalLeakage"

```
class NoInertgasSupply(Cause):
    equivalent_to = [Cause &
                     causeInvolvesEquipmentEntity.some(InertgasBlanketingEntity) &
                     causeInvolvesSubstance.some(hasSpecificTask.some(InertGas)) &
                     isCauseOfDeviation.some(NoFlow)]
```

Listing 4: Class definition of "NoInertgasSupply"

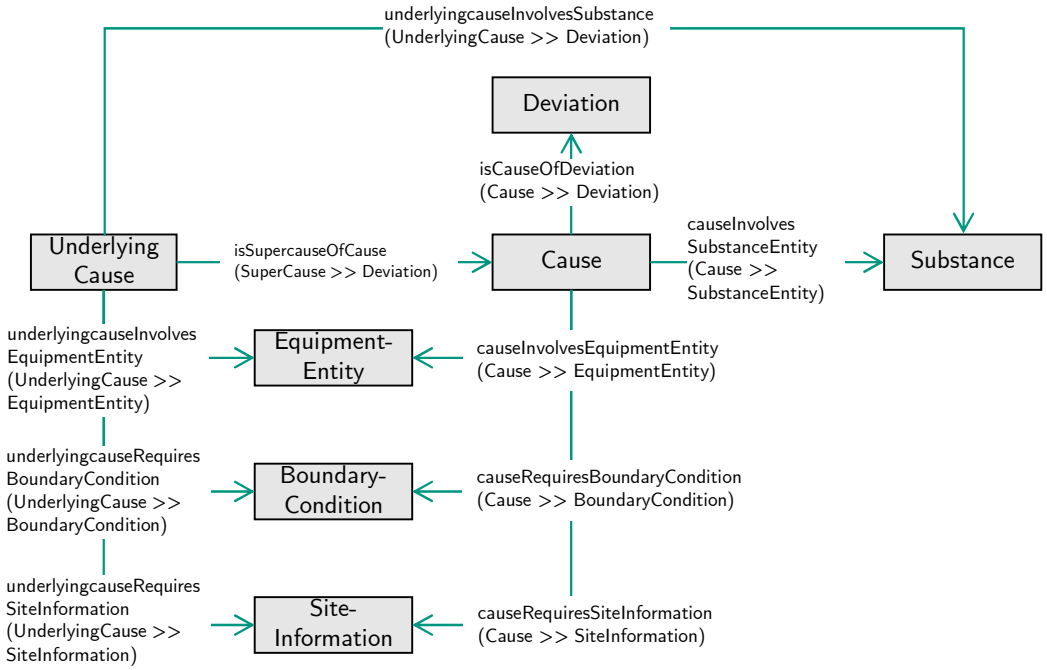


Figure 52: Concept map of cause-related semantic relationships

5.8.3 Semantic relationships of effects and consequences

In Addition to the relationships between **Deviation**, **Causes**, **Effects,** and **Consequences** described in Section 5.2.3, there are relationships between **Effect**, **Consequence** concepts, and **EquipmentEntity**, **Substance**, **BoundaryCondition**, and **SiteInformation** (cf. Figure 53). These extended relationships enable the modeling of scenario components in sufficient detail. As described in Sections 5.2.3 and Table 9, a distinction is made between effects and consequences. The **Effect** concept describes intermediate events that lie in the event sequence between the fault event (deviation) and hazardous/operability event (consequence) (cf. Figure 28). Through the relationships to **EquipmentEntity** and **Substance,** all subsumed classes and relationships, such as **Apparatus**, **Fixture, HazardClass,** and **StabilityReactivityInformation,** are considered implicitly (cf. Section 5.4, Section 5.5). There are relationships between the **SiteInformation** and **Effect** or **Consequence** concepts because site information (e.g., ambient temperatures) can influence effects and consequences (cf. Figure 53). Furthermore, consequences can lead to subsequent consequences, modeled by the isSubsequentConsequence relationship (marked in Figure 53 by the reference to itself). The relationships in Figure 53 are optional and allow the modeling of scenario components, but not all relationships are required for modeling specific scenario components. Inverse properties are intentionally not modeled (cf. Section 5.8.1).

For instance, Listing 5 shows an excerpt of the class definition “FluidCirculatesInsidePump”, which involves object properties (relationships) with the equipment entity, deviation, cause, intended function, and mode of operation. Listing 6 shows the class definition of the consequence “DangerOfBleve”, which involves the effect “FluidCirculatesInsidePump” and has relationships to the equipment entity and the cause “DeadHeadingOfPump”. These examples of scenario components are intended to illustrate the relationships in Figure 53; complete scenarios are shown in detail in Sections 5.9.1 and 0.

```
class FluidCirculatesInsidePump(Effect):
    equivalent_to = [Effect &
                     isEffectOfDeviation.some(NoFlow) &
                     effectImpliedByCause.some(DeadHeadingOfPump | PumpingAgainstPolymerizedLine) &
                     effectInvolvesEquipmentEntity.some(
                                  hasMaterialTransferEquipment.some(CentrifugalPump) &
                                  hasOperationMode.some(DeliverConstantVolumeFlow))]
```

Listing 5: Excerpt of the effect class definition "FluidCirculatesInsidePump"

```
class DangerOfBleve(Consequence):
    equivalent_to = [Consequence &
                     isConsequenceOfEffect.some(FluidCirculatesInsidePump) &
                     consequenceImpliedByCause.some(DeadHeadingOfPump) &
                     consequenceInvolvesEquipmentEntity.some(
                                  hasMaterialTransferEquipment.some(CentrifugalPump))]
```

Listing 6: Class definition of the consequence "DangerOfBleve"

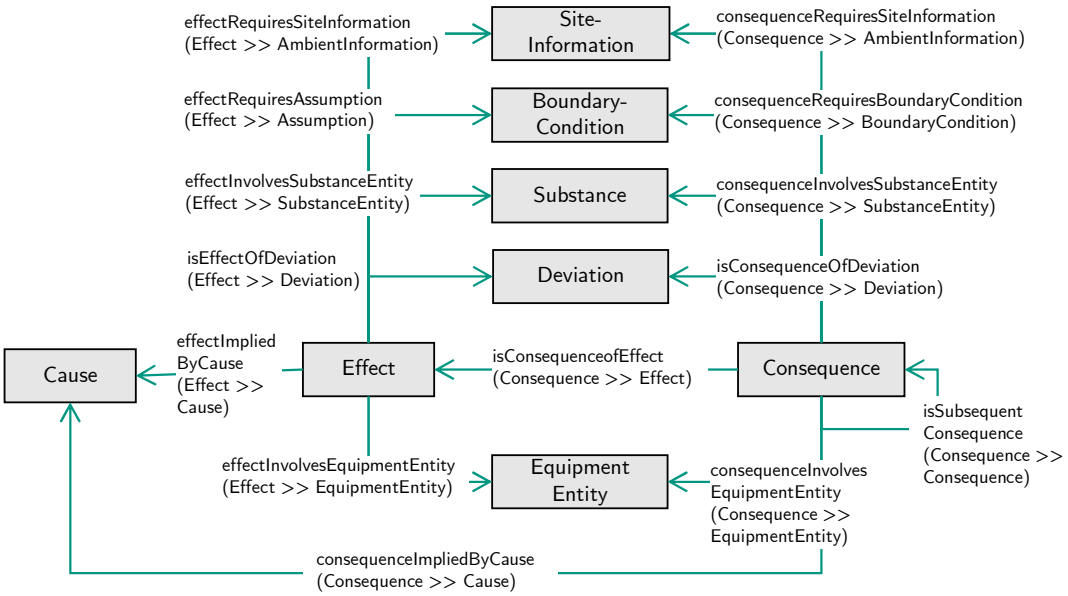


Figure 53: Concept map of effect and consequence-related semantic relationships

5.8.4 Semantic relationships of likelihood, severity, and risk

The risk assessment and raking of scenarios are crucial for determining suitable safeguards and follow-up measures. The risk category is determined based on likelihood and severity. The relationships between the **RiskCategory** and the **LikelihoodCategory** are represented via the involvesLikelihood relationship, while involvesSeverity relates the **SeverityCategory** and **RiskCategory** (cf. Figure 54). Listing 7 shows the class definition of risk category **A** as an example based on the risk matrix from Figure 6. The likelihood of scenarios is cause-related, and the relationships likelihoodInvolvesCause and likelihoodInvolvesUnderlyingCause exist between **Likelihood**, **UnderlyingCause**, and **Cause**. Additionally, the likelihood of a scenario depends on the equipment entity, boundary conditions, and site information expressed by the relationships

likelihoodInvolvesEquipmentEntity, likelihoodRequiresBoundaryCondition, and likelihoodInvolvesSiteInformation. The severity refers to the consequence of the scenario, which is indicated by the relationship isSeverityOfConsequence. Additionally, substance properties, especially the GHS hazard classes (cf. Figure 33), are required to determine the severity. Hence, the relationship severityInvolvesSubstance exists between the **SeverityCategory** and **Substance** concept. Additionally, there are relationships between the severity and boundary conditions (severityRequiresBoundaryCondition) and the equipment entity (severityInvolvesEquipmentEntity). For example, Listing 8 shows a snippet of the **Severe** class definition that involves the boundary condition **PersonnelPresentInTheNearField** and its other components.

```
class A(RiskCategory):
    equivalent_to = [RiskCategory &
                     ((involvesLikelihood.some(VeryLikely) & involvesSeverity.some(Catastrophic)) |
                     (involvesLikelihood.some(Likely) & involvesSeverity.some(Catastrophic)) |
                     (involvesLikelihood.some(VeryLikely) & involvesSeverity.some(Severe)))]
```

Listing 7: Class definition of risk category "A"

```
class Severe(SeverityCategory):
    equivalent_to = [SeverityCategory &
                     severityInvolvesEquipmentEntity.some(StorageTankEntity) &
                     severityInvolvesSubstance.some(
                                  hasHazardClass.some(FlammableLiquidCategory1 |
                                                      FlammableLiquidCategory2)) &
                     severityRequiresBoundaryCondition.some(PersonnelPresentInTheNearField) &
                     isSeverityOfConsequence.some(LossOfPrimaryContainment)]
```

Listing 8: Class definition of the severity category "Severe"

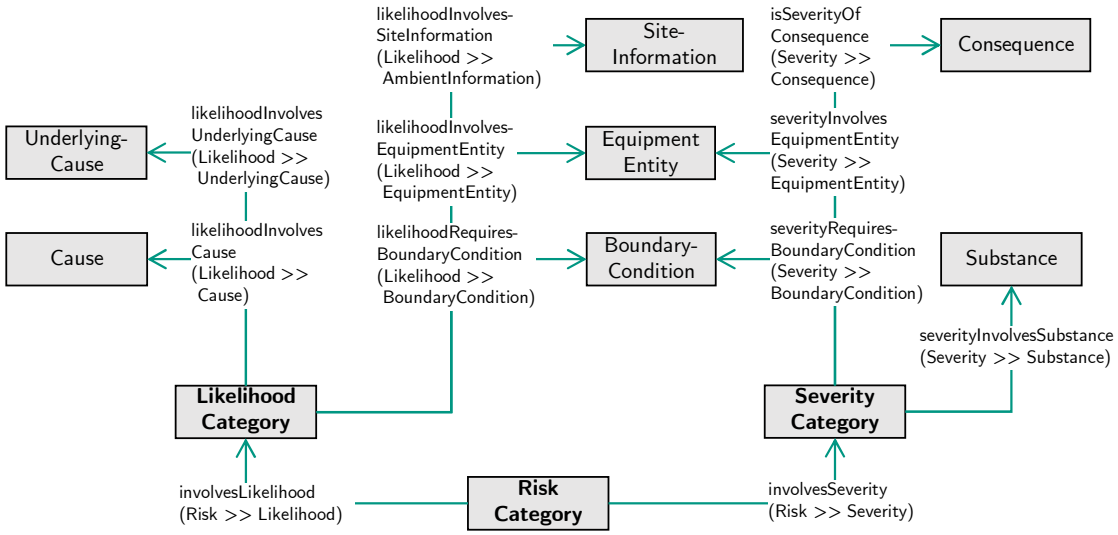


Figure 54: Concept map of likelihood, severity, and risk-related semantic relationships

5.8.5 Semantic relationships of safeguards

Safeguards aim to interrupt the event sequence of scenarios or mitigate their consequences (cf. Section 5.8.4). Since the scenario components stand for different events in the event sequence,

they can be interrupted at different points depending on the scenario. In addition, not every scenario necessarily consists of all scenario components. For example, underlying causes are not necessarily assigned to scenarios; accordingly, it is not sufficient to relate safeguards only to the underlying cause, representing the first event in the event sequence (cf. Figure 28). Due to the combinatorial variety of scenario components, safeguards can refer to all scenario components. Besides, the extent and availability of the safeguards depend on the risk of the scenario. Accordingly, semantic relationships must be established between all individual scenario components and the **Safeguard** concept. In Figure 55, a concept map of the relationships of the **Safeguard** concept is shown. In addition to the core semantic relationships, there are also additional relationships to equipment entity, substance, boundary condition, and site information. Concerning the concept map, safeguards that aim to prevent (underlying) causes are expressed via the relation safeguardPreventsUnderlyingCause and safeguardPreventsCause that relate the **Safeguard** and the **UnderlyingCause** and **Cause** concept. Additionally, safeguards can imply follow-up safeguards, represented via the impliesSafeguard relationship. Furthermore, via the safeguardPreventsEffect relationship effects are related to safeguards. The **Safeguard** concept is related to **Consequence**s via the relationship safeguardMitigatesConsequence. Additionally, safeguards affect the process deviations expressed by the relationship safeguardOfDeviation. Since safeguards depend directly on the substance properties, there are relationships to the **Substance** concept via the safeguardInvolvesSubstance property. Likewise, safeguards depend on the equipment entities, expressed by the relation safeguardInvolvesEquipmentEntity. In addition, safeguards depend on the risk class of the scenario, and the extent and quality requirements of safeguards depend on the risk class. The relation safeguardDependsOnRiskCategory (cf. Figure 55) is used to represent this context.

Listing 9 shows an excerpt of the class definition of the safeguard "IncreaseClosingTimeOfValve" to prevent a "Waterhammer", which involves the equipment entity **ConnectionPipeEntity**. In Listing 10, the safeguard "ThermalProtectionTrip" is intended to protect a pump or compressor against a too high temperature involving the effect of "Overheating". In Listing 11, the safeguard "IncreaseSafetyIntegrityLevel" is intended to prevent the underlying cause "LevelIndicatorControllerFailure" in conclusion with the risk category **B** or **C**. Regarding the selection of safeguards, there are always several options, and it is a matter of judgment and depends on the plant manager or responsible project manager.

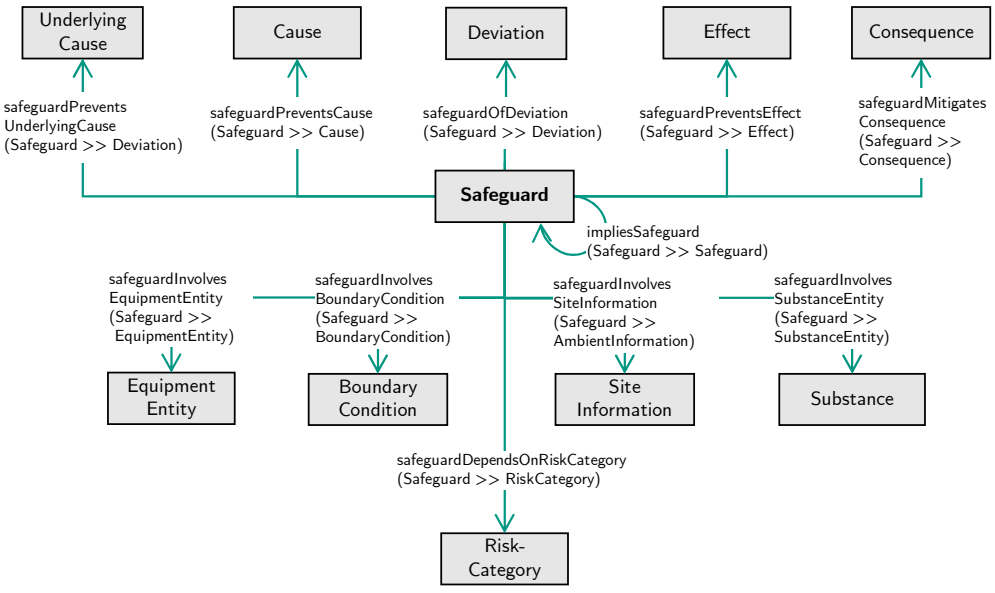


Figure 55: Concept map of safeguard-related semantic relationships

```
class IncreaseClosingTimeOfValve(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity(ConnectionPipeEntity) &
                     safeguardPreventsCause.some(Waterhammer)]
```

Listing 9: Excerpt of the class definition "SwingCheckValve"

```
class ThermalProtectionTrip(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity(PumpEntity | CompressorEntity) &
                     safeguardOfDeviation.some(HighTemperature) &
                     safeguardPreventsEffect.some(Overheating)]
```

Listing 10: Excerpt of the class definition "ThermalProtectionTrip"

```
class IncreaseSafetyIntegrityLevel(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardDependsOnRiskCategory.some(C | B) &
                     safeguardPreventsUnderlyingCause.some(LevelIndicatorControllerFailure)]
```

Listing 11: Excerpt of the class definition "IncreaseSafetyIntegrityLevel"

5.9 Modeling of application-oriented ontology classes

The theoretical framework of the ontology presented in the concept maps is the foundation to create causal models representing the HAZOP scenarios. The concept maps represent informal knowledge models, while the example class definitions represent formal concepts of the ontology. These informal knowledge models are formalized using the ontology language OWL 2 DL.

5.9.1 Examples of formal class definitions

The developed ontology is based on basic set theory and formal logic, and class definitions can be expressed using description logic (DL) syntax. An example scenario will be presented based on formal definitions of the scenario components in the following. The scenario involves an atmospheric storage tank where an external fire leads to abnormal evaporation, impermissible overpressure, and a fracture resulting in a loss of primary containment (LOPC). An LOPC is defined as an "unplanned or uncontrolled release of material from primary containment, including non-toxic and non-flammable materials" (cf. CCPS, n.d.). Hence, the release of hazardous substances and non-hazardous substances are assigned to the term LOPC. Generally, a class definition consists of the parent class and, on the other hand, of restrictions via data or object properties. First, the scenario component **UnderlyingCause** is considered. The class expression is "ExternalFire" (cf. Eq. (2)), where cause and underlying cause are related using the object properties isUnderlyingCauseOfCause (cf. Figure 31) and underlyingcauseRequiresBoundaryCondition (cf. Figure 53). This class definition translates as an "ExternalFire" is the underlying cause of the cause "AbnormalHeatInput". The corresponding formal definition can be expressed using the DL syntax as follows:

$$\begin{aligned} \mathit{ExternalFire} \equiv\ & \mathit{UnderlyingCause} \sqcap \\ & \exists \mathit{isUnderlyingcauseOfCause}.\mathit{AbnormalHeatInput} \sqcap \\ & \exists \mathit{underlyingcauseRequiresBoundaryCondition}.\mathit{ExternalFirePossible}. \end{aligned} \tag{2}$$

The following scenario component within the causal chain is the **Cause** "AbnormalHeatInput" (cf. Eq. (3)). The example describes the cause "AbnormalHeatInput", which involves the apparatus **AtmosphericStorageTank** and the deviation **HighTemperature**. The class is equivalent to the **Cause** and provided with the existential restrictions (∃) in the form of the object property expressions causeInvolvesEquipmentEntity and isCauseOfDeviation. The existential restriction hasApparatus is embedded within the causeInvolvesEquipmentEntity. The class definition "AbnormalHeatInput" is intended to explain the principal procedure for formulating formal class definition and is not necessarily exhaustive. If the class definition is extended, required restrictions are appended to the class definition with an OR operator.

$$\begin{aligned} \mathit{AbnormalHeatInput} \equiv\ & \mathit{Cause} \sqcap \\ & \exists \mathit{causeInvolvesEquipmentEntity}.(\exists \mathit{hasApparatus}.\mathit{AtmosphericStorageTank}) \sqcap \\ & \exists \mathit{isCauseOfDeviation}.\mathit{HighTemperature} \end{aligned} \tag{3}$$

Within the following example, the effect "AbnormalEvaporation" is described (cf. Eq. (4)). It is equivalent to the **Effect** class and has multiple existential restrictions. It involves the object property expression effectInvolvesEquipmentEntity. Besides, the **Cause** from Eq. (3) is involved within this **Effect** via the object property expression effectImpliedByCause. This definition translates as the effect "AbnormalEvaporation", takes place in an "AtmosphericStorageTank" that contains a liquid or multiphase substance involving an "AbnormalHeatInput", which can occur within the intended functions "Filling" or "Storing":

$$\begin{aligned} \mathit{AbnormalEvaporation} \equiv\ & \mathit{Effect} \sqcap \\ & \exists \mathit{effectImpliedByCause}.\mathit{AbnormalHeatInput} \sqcap \\ & \exists \mathit{effectInvolvesEquipmentEntity}.(\exists \mathit{hasApparatus}.\mathit{AtmosphericStorageTank} \sqcap \\ & \qquad \exists \mathit{hasIntendedFunction}.(\mathit{Filling} \sqcup \mathit{Storing})) \sqcap \\ & \exists \mathit{effectInvolvesSubstance}.(\exists \mathit{hasStateOfAggregation}.(\mathit{Liquid} \sqcup \mathit{Multiphase})) \end{aligned} \tag{4}$$

The follow-up effect of "AbnormalEvaporation" is "PressureExceedingDesignPressure" (cf. Eq. (5)) with the subsequent deviation "Fracture". The method for inferring the deviation **HighPressure** based on the effect "AbnormalEvaporation" is described in detail in Section 5.13.

$$\begin{aligned} \mathit{PressureExceedingDesignPressure} \equiv\ & \mathit{Effect} \sqcap \\ & \exists \mathit{effectInvolvesEquipmentEntity}.(\exists \mathit{hasApparatus}.\mathit{AtmosphericStorageTank}) \sqcap \\ & \exists \mathit{isEffectOfDeviation}.\mathit{HighPressure} \end{aligned} \tag{5}$$

In the following example, the consequence "LossOfPrimaryContainment" is described (cf. Eq. (6)). It involves an existential restriction with the object property expression isConsequenceOfEffect. This definition translates as the "LossOfPrimaryContainment" is a consequence of a "Fracture" of equipment involving the apparatus "AtmosphericStorageTank". Eq. (6) shows only an excerpt of the class definition "LossOfPrimaryContainment", and the complete class definition is shown in Appendix N.

$$\mathit{LossOfPrimaryContainment} \equiv \mathit{Consequence} \sqcap \exists \mathit{isConsequenceOfEffect}.\mathit{Fracture} \tag{6}$$

As a safeguard against the "PressureExceedingDesignPressure", the safeguard "EmergencyPressureReliefValve" is defined in Eq. (7). It involves the object property expression safeguardMitigatesEffect, and the **Effect** "PressureExceedingDesignPressure":

$$\begin{aligned} \mathit{EmergencyPressureReliefValve} \equiv\ & \mathit{Safeguard} \sqcap \\ & \exists \mathit{safeguardMitigatesEffect}.\mathit{ImpermissibleOverpressure} \sqcap \\ & \exists \mathit{safeguardInvolvesUnderlyingCause}.\mathit{ExternalFire} \sqcap \\ & \exists \mathit{safeguardInvolvesEquipmentEntity}.(\\ & \qquad \exists \mathit{hasApparatus}.\mathit{AtmosphericStorageTank}) \end{aligned} \tag{7}$$

This formal definition translates as the safeguard emergency pressure relief valve that prevents impermissible overpressure of a storage tank in case of an external fire. In addition to using object properties (e.g., safeguardMitigatesEffect), data properties can also be used within the OWL 2 ontology language, which allows using numerical values. For example, the German technical regulations for hazardous substances (TRGS 509) define limits for the flashpoint of liquid substances above which minimum separation distances between storage tanks must be kept. This regulation can be represented in a safeguard as follows:

$$\begin{aligned} \mathit{DistanceBetweenTanks} \equiv\ & \mathit{Safeguard} \sqcap \\ & \exists \mathit{safeguardInvolvesEquipmentEntity}.\mathit{StorageTankEntity} \sqcap \\ & \exists \mathit{safeguardInvolvesSubstance}.(\exists \mathit{substanceHasFlashpointInKelvin}.\leq 328.15). \end{aligned} \tag{8}$$

Another essential concept is the disjoint classes axiom, which means that concepts do not overlap. According to Rudolph, disjointness information is often neglected, which can be disadvantageous (cf. Rudolph, 2011, pp. 106). For instance, **Cause** and **UnderlyingCause** do not overlap under any circumstances because they stand for fundamentally different scenario components. This relationship can be illustrated using the DL syntax as follows:

$$\mathit{Cause} \sqsubseteq \neg\ \mathit{UnderlyingCause}. \tag{9}$$

When modeling the classes, the order of operations must be considered. For instance, AND ($\sqcap$) has higher precedence than OR ($\sqcup$) while NOT has higher precedence compared to AND ($\neg$). Accordingly, parentheses should be used to prevent logic errors.

5.9.2 Event sequence within scenarios

Based on the semantic relationships between the HAZOP core concepts (cf. Section 5.2.3), an extended causation model was derived, representing the event sequence from which the scenario results. These chains of events represent scenarios that are also represented in HAZOP worksheets. For the systematic modeling of the scenarios, complete scenarios were decomposed into their components within this research work to model them with the help of the provided concepts. The concepts for modeling causal chains include underlying causes, causes, deviations, effects, and consequences. In addition, the concepts of likelihood, severity, risk, and safeguard are available to complete the overall scenarios. Scenarios and their components have been modeled based on existing HAZOP worksheets, expert interviews, and technical literature. The corresponding class definitions are shown in the Appendix G to Appendix P. The underlying schematic model assumptions behind the causal chains of the scenarios are:

- causes can have different independent underlying causes,
- deviations can have different independent causes,
- deviations and causes are needed to determine effects, while specific effects depend only on deviations,
- multiple different effects can arise from one deviation,
- an effect may lead to different consequences,
- consequences depend directly on effects,
- consequences can lead to subsequent consequences (consequence sequence).

The distinction between underlying causes/causes and effects/consequences allows a more precise description of causal relationships within scenarios (cf. Section 5.3). In addition, monocausal and polycausal interrelationships can be better distinguished. Due to this approach, scenarios can be dynamically generated based on the scenario components and are not statically modeled in their entirety. Since the scenario components are semantically related, new scenarios can be created by adding individual scenario components. In a static consideration of scenarios,

every potential scenario would have to be modeled explicitly, which would not be scalable due to the combinatorial diversity of scenario components. The interaction of scenario components in the representation of scenarios will be illustrated in the following employing examples. These examples are intended to provide an overview of the scenarios that can be modeled. Hence, they are not necessarily exhaustive.

An excerpt of multiple event sequences will be illustrated through examples involving a centrifugal pump (cf. Figure 56). Operating errors or failure of the control loop can cause a pump's discharge to be closed or blocked, which leads to a deadhead (cf. "DeadHeadingOfPump", Figure 56), which is characterized by the process deviations 'no flow' and 'high temperature'. This can lead to overheating, or the pump could run completely isolated. While overheating can lead to a pump breakdown, a completely blocked-in centrifugal pump can lead to a boiling liquid expanding vapor explosion (BLEVE). With centrifugal pumps, operation below the minimum flow rate can increase temperature, leading to overheating and pump breakdown. A breakdown does not necessarily imply an LOPC, only that the pump is not working due to a malfunction, failure, or mechanical issue. Another operability issue would be an insufficient net positive suction head (NPSH) value, which leads to a too low suction pressure, which can cause cavitation and pump breakdown. Furthermore, a 'low flow' deviation could have an impeller issue as a cause, leading to a poor pump performance without further safety-related consequences.

Safeguards aim to interrupt the event sequence and thus prevent a hazardous event or operability issue (cf. Figure 28). The entire scenario must always be considered in full to model individual components of the scenarios. Individual scenario components stand in context with the other components and cannot be considered in isolation. Hence, the sequence of events of scenarios must be considered (cf. Figure 28). Figure 57 shows selected scenarios that can occur in association with a storage tank. A production worker could mistakenly open the drain valves of the storage tank (cf. "DrainValveInadvertentlyOpened", Figure 57), which could have the underlying causes "HumanError", "FaultyOperatingInstruction", or "InsufficientTraining" on the one hand and lead to the deviations **ElsewhereFlow** and **LowPressure** on the other hand. Assuming the substance in the vessel is liquid, the deviation **ElsewhereFlow** could lead to a pool formation and an LOPC.

Furthermore, by draining liquid without a vent valve, nitrogen blanketing, or other pressure control, a vacuum could build up in the storage tank, which in the worst case would result in a "LossOfMechanicalIntegrity" and a "LossOfPrimaryContainment".

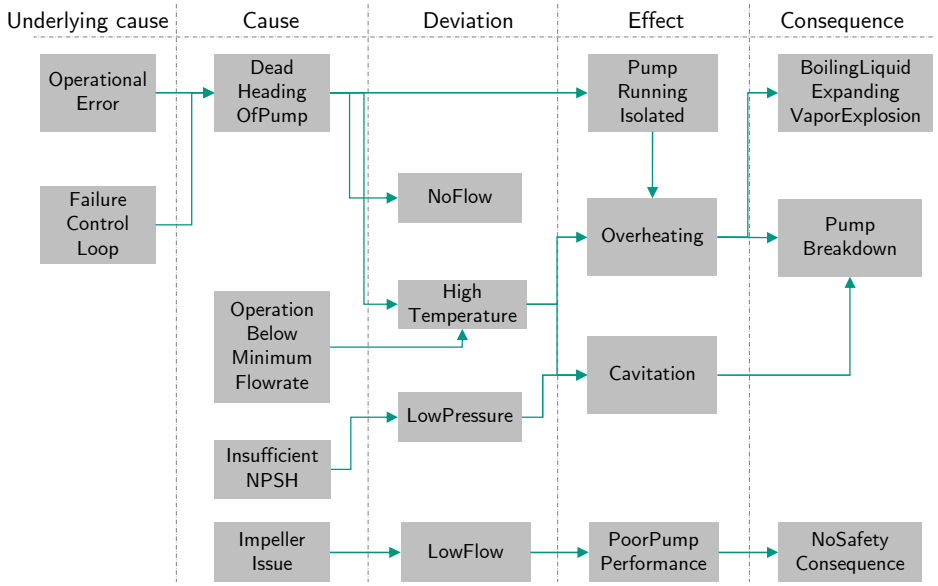


Figure 56: Partial cause-effect chain concerning a centrifugal pump

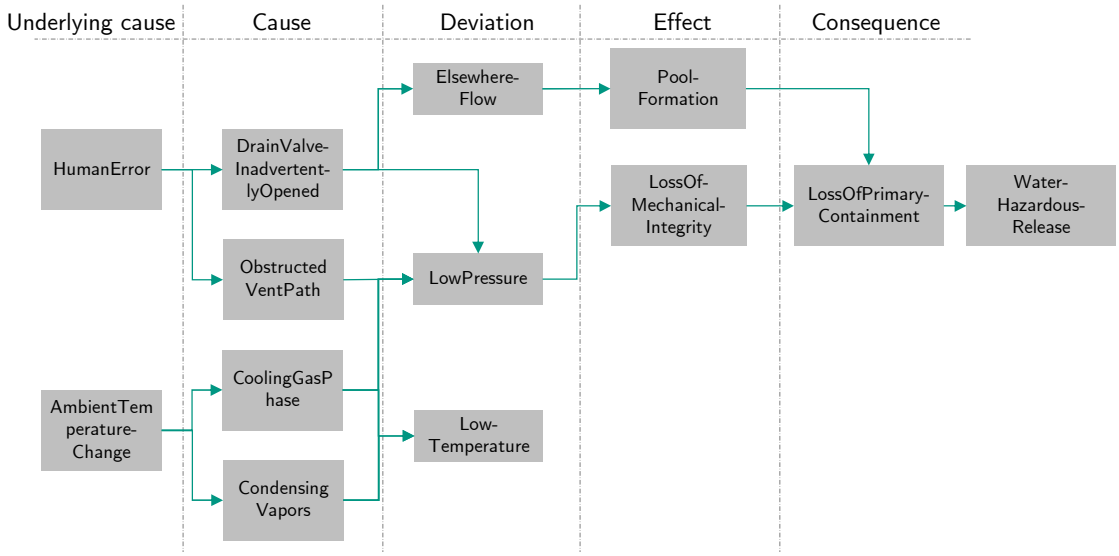


Figure 57: Partial cause-effect chain concerning a storage tank

If the content of the tank is hazardous to water, a "WaterHazardousRelease" can be assumed (cf. Figure 57). Besides, an obstructed vent path, the cooling of the gas phase, and condensing vapors within the storage tank may lead to the **LowPressure** deviation with further consequences.

5.9.3 Determination of the likelihood, severity, and risk

The risk of a scenario depends on the likelihood of the cause and the severity of the consequence. For example, if there is a low likelihood of occurrence, according to VDI 2180-1, a *basic process control system protection layer* (BPCS PL) as a process control measure that is checked regularly is sufficient as a safeguard. The relationships that are necessary to represent the relationships between causes and likelihood are shown in Figure 54. Generally, a distinction is made between the likelihood categories (cf. Figure 6):

- F0; very likely; $>$ 1 per year,
- F1; likely; 10^{-1} - 1 per year,
- F2; occasional; 10^{-2} - 10^{-1} per year,
- F3; possible; 10^{-3} - 10^{-2} per year,
- F4; unlikely; 10^{-4} - 10^{-3} per year,
- F5; very unlikely; 10^{-5} - 10^{-4} per year.

There are dependencies between the scenario components **UnderlyingCause**, **Cause**, and the equipment to determine the likelihood of a scenario. Examples of scenario components that determine different likelihood classes are shown in Table 14. The cause "InsufficientThermalBreathing" has the likelihood **VeryLikely** since it can happen any time the weather changes (cf. Table 14, Pos. 9). A "MechanicalFailureOfSupport" is **Unlikely** because it is rare. In the case of a "DangerOfSeismicActivity" (environmental danger, Table 13), the likelihood of the scenario increases to the likelihood category **Possible**. The severity is identified based on different scenario components (cf. Table 15). A distinction is made between the categories (cf. Figure 6):

- S0; catastrophic,
- S1; severe,
- S3; serious,
- S4; significant,
- S5; minor.

Table 14: Example relationships to other concepts and rationale for identifying the likelihood of causes

Pos.	Equipment	Site information	Deviation	Underlying Cause	Cause	Likelihood category	Event p.a.	Likelihood	Source
1	Pump	-		-	PumpSealFailure	F2	0.01-0.1	Occasional	(CCPS, 2001), pp. 71
2	Pump (Centrifugal)	-		SuddenlyStop-pingPump	-	F2	0.0554	Occasional	(OREDA, 2002), pp. 180
3	Pump (Centrifugal)	-	High-Vibration	-	-	F2	0.0185	Occasional	
4	Pump	-	-	-	PumpFailure	F1	-	Likely	(IVSS, 2020), pp. 58
5	-	-		ExternalFire	-	F2	0.001-0.1	Occasional	(CCPS, 2001), pp. 71
6		-	-	BrokenHose	-	F1	0.01-1	Likely	(CCPS, 2001), pp. 71
7	StorageTank	DangerOfSeis-micActivity	-	-	MechanicalFailure OfSupport	F3	-	Possible	own consideration
8	StorageTank	-	-	-	MechanicalFailure OfSupport	F4	-	Unlikely	own consideration
9	StorageTank	-	-	-	InsufficientThermal-Breathing	F0	-	VeryLikely	own consideration

Table 15: Example relationships to other concepts for identifying the severity of consequences

Pos.	Equipment	Boundary Condition	Substance hazard class	Consequence	Severity category	Severity	Source
1	-	-	AspirationHazardCategory1	LossOfPrimaryContainment	S1	Severe	(IVSS, 2020), pp. 55
2	-	-	SkinCorrosionIrritationCategory1		S1	Severe	
3	-	-	PyrophoricGasCategory1		S0	Cata-strophic	
4	-	-	-	ProductionDowntime	S4	Minor	own considera-tion
5	Pump	-	-	PumpBreakdown	S4	Minor	
6	-	-	SpecificTargetOrganToxicityRepeatedEx-posureCategory3	LossOfPrimaryContainment	S2	Serious	

The boundary condition **SeveralPeoplePresentInTheNearField** increases the severity of releasing a hazardous substance combined with a fire or explosion hazard (cf. Table 15, Pos. 1). The hazard classes of the substances are also essential while evaluating the severity of scenarios. The release of a substance with the hazard classes **AspirationHazardCategory1** or **SkinCorrosionIrritationCategory1** leads to the severity category **Severe** (cf. Table 15, Pos. 2-3). Based on the risk matrix (cf. Figure 6), the likelihood and severity categories directly result in the risk class. The formal definitions of the relationships between the concepts **Likelihood**, **Severity**, and **Risk** are shown in Appendix Listing O-1. The likelihood class definitions are shown in Appendix Listing O-1 and Appendix Listing O-2, the severity class definitions in Appendix Listing O-3. The risk class definitions transferred from the risk matrix are shown in Appendix Listing O-4. For instance, the severity of consequences, such as "ProductionDowntime", or "PumpBreakdown", are ranked with the severity category **Minor** (cf. Table 15, Pos. 5-6).

5.9.4 Programmatic creation of ontology classes and example class definitions

The Python package Owlready2 was used to programmatically create the ontology in an object-oriented manner. The syntax of Owlready2 is based on the Manchester syntax (cf. Table 5). To illustrate the procedure for defining ontology classes, a Python-based class definition using Owlready2, including syntactic descriptions of components, is shown in Figure 58. It defines the class "DrainValveInadvertentlyOpened", which belongs to the parent class **Cause**. The definition is made with the *equivalent class axiom*, which means that the class corresponds to a cause with the restrictions 1-4 (cf. Figure 58). Each restriction is provided with the logical operator & (AND), which means that all restrictions must be fulfilled. The reasoner concludes that the cause "DrainValveInadvertentlyOpened" is valid only if all four conditions are met. In the first restriction, the object property isCauseOfDeviation is expressed as an existential restriction (cf. Section 3.8.3) to describe the relationship of the class to the deviations **ElsewhereFlow** OR (|) **LowPressure**. Hence, the unintended opening of a drain valve is the cause of the deviation 'low pressure' or 'elsewhere flow'. The second restriction involves a bottom drain valve required for the cause to be plausible. This restriction is expressed as an embedded relationship since the relationship hasInstrumentation is embedded within the relationship causeInvolvesEquipmentEntity. These embeddings mean no specific **EquipmentEntity** is defined but that every equipment with a drain valve and the operation mode **ModeIndependent** is considered. The third restriction describes the state of aggregation through the relationship hasStateOfAggregation, which can be both **Liquid** and **Multiphase**. The involved state of aggregation is implemented as an embedded restriction.

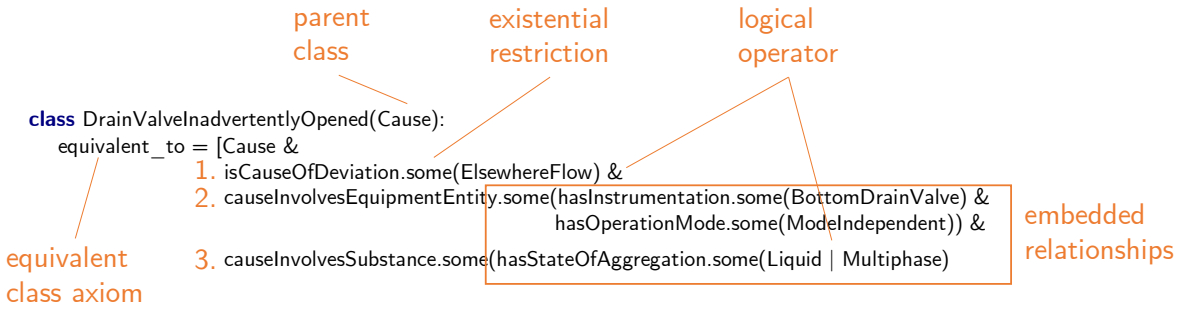


Figure 58: Description of the components of the formal class definitions of the ontology

```
class ExcessiveFluidWithdrawal(Cause):
    equivalent_to = [Cause &
                     causeInvolvesEquipmentEntity.some(StorageTankEntity) &
                     isCauseOfDeviation.some(LowPressure)]
ExcessiveFluidWithdrawal.comment = ["The issue of depressurization and collapse of tank is addressed",
"https://www.aiche.org/resources/publications/cep/2019/december/protect-tanks-overpressure-and-vacuum", "API
2000"]
```

Listing 12: Programmatic definition of a class annotation

Furthermore, fundamental restrictions, such as the disjointness axioms between the core classes **UnderlyingCause**, **Cause**, **Deviation**, **Effect**, **Consequence**, **Safeguard**, **Likelihood**, **Severity**, and **Risk,** must be considered during ontology development (cf. Eq. (9)). As a result, no individual of one class can belong to another class. In addition, specific class definitions are provided with comments or explanations about the meaning, synonyms, or classification of terms using annotations (cf. Section 3.8.2). For instance, in Listing 12, an annotation in the form of a comment is associated with the class "ExcessiveFluidWithdrawal". The annotation provides resources to further information. The programmatic creation of the ontology allows it to be easily extended and unit tests to be performed. During ontology development, unit tests were created for individual ontology classes. Appendix Listing S-1 shows the example implementation of unit tests. Within the unit tests, conclusions were drawn regarding the correctness of individual ontology class definitions. Regularly running the unit tests during development and evaluating the inference drawn by the reasoners is an effective way to verify class definitions. Besides, unit tests allow tracking the influence of local changes on global class definitions.

5.9.5 Importance of the boundary conditions and site information

Within this section, examples regarding the use of **BoundaryCondition**s and **SiteInformation** are described to underline their importance. Boundary conditions are defined for each equipment entity (cf. Figure 49). For instance, the **Cause** "EntrainedAir", which can occur within a pump, requires the **IntroductionOfAir** to be inferred (cf. Listing 13). In the following example, the more complex class definition of the consequence "RiskOfExplosiveAtmosphere" is considered, consisting of four blocks in an OR relation (cf. Listing 14). The division of the class definitions into blocks increases the readability, primarily if a class definition consists of multiple OR conditions. In principle, an explosive atmosphere can form outside and inside the enclosure. An explosive atmosphere does not necessarily imply an explosion. The first OR block of the class definition consists of the consequence "LossOfPrimaryContainment", the GHS hazard classes: **PyrophoricGasCategory1**, **FlammableGasCategory1**, **FlammableGasCategory2**, **FlammableLiquidCategory1**, **FlammableLiquidCategory2**, **FlammableLiquidCategory3**, **AerosolCategory1**, **AerosolCategory2**. Moreover, the boundary condition **SufficientOxygenAvailable** is implied because the plant is located outside. In the second block of the class definition, the stability or reactivity information "FormsExplosiveMixtureWithAir" is considered instead of the hazard classes, extracted from the safety data sheet. This restriction is embedded within the object property consequenceInvolvesSubstance. In the third block, a situation is modeled where a tank or pressure vessel is inadequately inerted ("InsufficientInertization"), air enters, and there is a risk of an explosive atmosphere in the enclosure. The last part of the class definition models the situation that an "InternalLeakage" of a substance with the stability or reactivity information "FormsExplosiveMixturesWithOxidizingAgents" leads to the risk of an explosive atmosphere. The described class definition shows how different situations can be represented within one class definition. In addition, boundary conditions are of decisive importance for an unambiguous class definition, especially in complex classes.

```
class EntrainedAir(Cause):
    equivalent_to = [Cause &
                     causeInvolvesEquipmentEntity.some(PumpEquipmentEntity) &
                     causeRequiresBoundaryCondition.some(IntroductionOfAir) &
                     isCauseOfDeviation.some(OtherThanComposition)]
```

Listing 13: Class definition function of "EntrainedAir"

```
class RiskOfExplosiveAtmosphere(Consequence):
    equivalent_to = [Consequence &
                ((consequenceRequiresBoundaryCondition.some(LocatedOutside | SufficientOxygenAvailable) &
                 consequenceRequiresBoundaryCondition.some(DelayedIgnition) &
                 isSubsequentConsequence.some(LossOfPrimaryContainment) &
                 consequenceInvolvesSubstance.some(hasHazardClass.some(FlammableLiquidCategory1 |
                                                                       FlammableLiquidCategory2 |
                                                                       FlammableLiquidCategory3 |
                                                                       FlammableGasCategory1 |
                                                                       FlammableGasCategory2 |
                                                                       PyrophoricGasCategory1 |
                                                                       AerosolCategory1 |
                                                                       AerosolCategory2)))
                |
                (consequenceRequiresBoundaryCondition.some(LocatedOutside | SufficientOxygenAvailable) &
                 isSubsequentConsequence.some(LossOfPrimaryContainment) &
                 consequenceRequiresBoundaryCondition.some(DelayedIgnition) &
                 consequenceInvolvesSubstance.some( hasStabilityReactivityInformation.some(FormsExplosiveMixtureWithAir)))
                |
                (isConsequenceOfEffect.some(InsufficientInertization) &
                  consequenceInvolvesEquipmentEntity.some(AtmosphericStorageTank | SettlingTankEntity) &
                 consequenceRequiresBoundaryCondition.some(IntroductionOfAir) &
                 consequenceInvolvesSubstance.some(hasHazardClass.some(FlammableLiquidCategory1 |
                                                                       FlammableLiquidCategory2 |
                                                                       FlammableLiquidCategory3 |
                                                                       FlammableGasCategory1 |
                                                                       FlammableGasCategory2 |
                                                                       PyrophoricGasCategory1 |
                                                                       AerosolCategory1 |
                                                                       AerosolCategory2)))
                |
                (consequenceImpliedByCause.some(InternalLeakage) &
                 consequenceInvolvesSubstance.some(
                         hasStabilityReactivityInformation.some(FormsExplosiveMixturesWithOxidizingAgents)))]
```

Listing 14: Class definition function of "FormationOfExAtmosphere"

The following example will present two insufficient class definitions and their effects on other scenario components. Two contextually correct but semantically inadequate class definitions for the cause classes "ThermalExpansion" and "Waterhammer" are shown in Listing 15. Both classes describe the causes of the 'high pressure' deviation within a pipeline. The **Cause** "ThermalExpansion" can be attributed to the **UnderlyingCause** "BlockedPipingAndHeatInput", while the "WaterHammer" can be attributed to "RapidlyClosingValve". From the class definitions "Waterhammer" and "ThermalExpansion", the erroneous conclusion is that:

$$\textit{ThermalExpansion} \equiv \textit{WaterHammer.} \tag{10}$$

Since both class definitions in Listing 15 share restrictions regarding the state of aggregation, the involved equipment entity, and the deviation, the reasoner equates the classes due to their intersections.

The inadmissible equating of classes leads to the following semantically correct but technically incorrect conclusions being drawn about the underlying causes:

- **Cause**: "ThermalExpansion" → **UnderlyingCause**: "BlockedPipingAndHeatInput", "RapidlyClosingValve",
- **Cause**: "WaterHammer" → **UnderlyingCause**: "BlockedPipingAndHeatInput", "RapidlyClosingValve".

The underlying cause "WaterHammer" is not a "BlockedPipingAndHeatInput", and a "RapidlyClosingValve" does not lead to a "ThermalExpansion". Since the heat input into the pipeline requires an external fire or solar radiation, the definitions can be completed using the boundary conditions **ExternalFirePossible** or **LocatedOutside**. The extension of the class definitions leads to distinguishable causes and correctly associated underlying causes. This type of incorrect class definition can be identified by unit tests in which all scenario components are evaluated one after the other to represent the entire scenario (cf. Section 5.9.4).

```
class ThermalExpansion(Cause):
    equivalent_to = [Cause &
                     causeInvolvesSubstance.some(hasStateOfAggregation.some(Liquid | Multiphase | Gaseous)) &
                     causeInvolvesEquipmentEntity.some(ConnectionPipeEntity) &
                     isCauseOfDeviation.some(HighPressure)]

class WaterHammer(Cause):
    equivalent_to = [Cause &
                     causeInvolvesSubstance.some(hasStateOfAggregation.some(Liquid) &  hasSpecificTask.some(ProcessMedium)) &
                     isCauseOfDeviation.some(deviation_onto.HighPressure) &
                     causeInvolvesEquipmentEntity.some(ConnectionPipeEntity)]

class BlockedPipingAndHeatInput(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(ThermalExpansion) &
                     underlyingcauseInvolvesEquipmentEntity.some(ConnectionPipeEntity)]

class RapidlyClosingValve(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(WaterHammer) &
                     underlyingcauseInvolvesEquipmentEntity.some(ConnectionPipeEntity)]
```

Listing 15: Insufficient class definitions of the causes "ThermalExpansion" and "Waterhammer" and the underlying causes "BlockedPipingAndHeatInput" and "RapidlyClosingValve"

```
class ThermalExpansion(Cause):
    equivalent_to = [Cause &
                     causeInvolvesSubstance.some(
                     hasStateOfAggregation.some(Liquid | Multiphase | Gaseous) &
                     causeInvolvesEquipmentEntity.some(ConnectionPipeEntity) &
                     causeRequiresBoundaryCondition.some(ExternalFirePossible | LocatedOutside) &
                     isCauseOfDeviation.some(HighPressure))]

class BlockedPipingAndHeatInput(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(ThermalExpansion) &
                     underlyingcauseRequiresBoundaryCondition.some(ExternalFirePossible | LocatedOutside)
                     underlyingcauseInvolvesEquipmentEntity.some(ConnectionPipeEntity)]
```

Listing 16: Enhanced class definition of "ThermalExpansion" and "BlockedPipingAndHeatInput"

5.10 Stages of development of ontology

Figure 59 visualizes the ontology development process considering different qualitative and quantitative aspects. Quantities such as the number of classes and properties are used in the process. Besides, the presence and the extent of domain-specific details are compared. In addition, selected ontology metrics were used to represent different development stages. The development status at the time of publications was chosen as development stages. Relative comparisons between characteristics are presented to show the development progress. The development stage with the highest absolute value is used as the benchmark. The presence of features is rated as 1 or 0 (cf. Risk assessment considered: 1, Figure 59, December 2019). The development stages were represented using the following characteristics:

- Quantitative characteristics
    - The relative number of classes: an indicator for the overall size of the ontology,
    - The relative number of object properties: an indicator for the overall interconnectedness of concepts,
    - The relative number of data properties: increases the depth of information
- Domain-specific content
    - The relative number of deviations: indicates the diversity of deviations,
    - Relative equipment detail: quantifies the extent of equipment related classes and proper
    - Relative level of detail substance: extent of considered substance properties,
    - Risk assessment considered: necessary for the appropriate determination of safeguards,
    - Site information considered: provides causes for specific scenarios,
- Ontology metrics
    - Relationship richness: quantifies the diversity of relationships (cf. Section 7.2.5),
    - Inheritance richness: a measure of the distribution of information (cf. Section 7.2.5).

At the beginning of the ontology development process (cf. Figure 59, April 2019), boundary conditions, risk assessment, and site information were not considered. The level of detail of the equipment and substances was rudimentary. Besides, the total number of classes and relationships was severely limited. Within a conceptual ontology (cf. Figure 59, December 2019), the risk assessment of scenarios has been considered, the number of available deviations has been extended, and a changed ontology structure results in strong changes in the ontology metrics. The use of ontology metrics to evaluate ontologies is described in detail in Section 7.2.5. As part of further development, the level of detail regarding the equipment and the total number of classes and object properties were significantly increased. The latest ontology (August 2021) has the highest value in most characteristics because it is the latest development stage. Due to further details within the class hierarchy regarding the equipment entities, the relationship richness is minimally lower than in the advanced ontology (August 2020). Additionally, the inheritance richness is lower than the advanced ontology (August 2020), suggesting that more details are present (cf. Section 7.2.5).

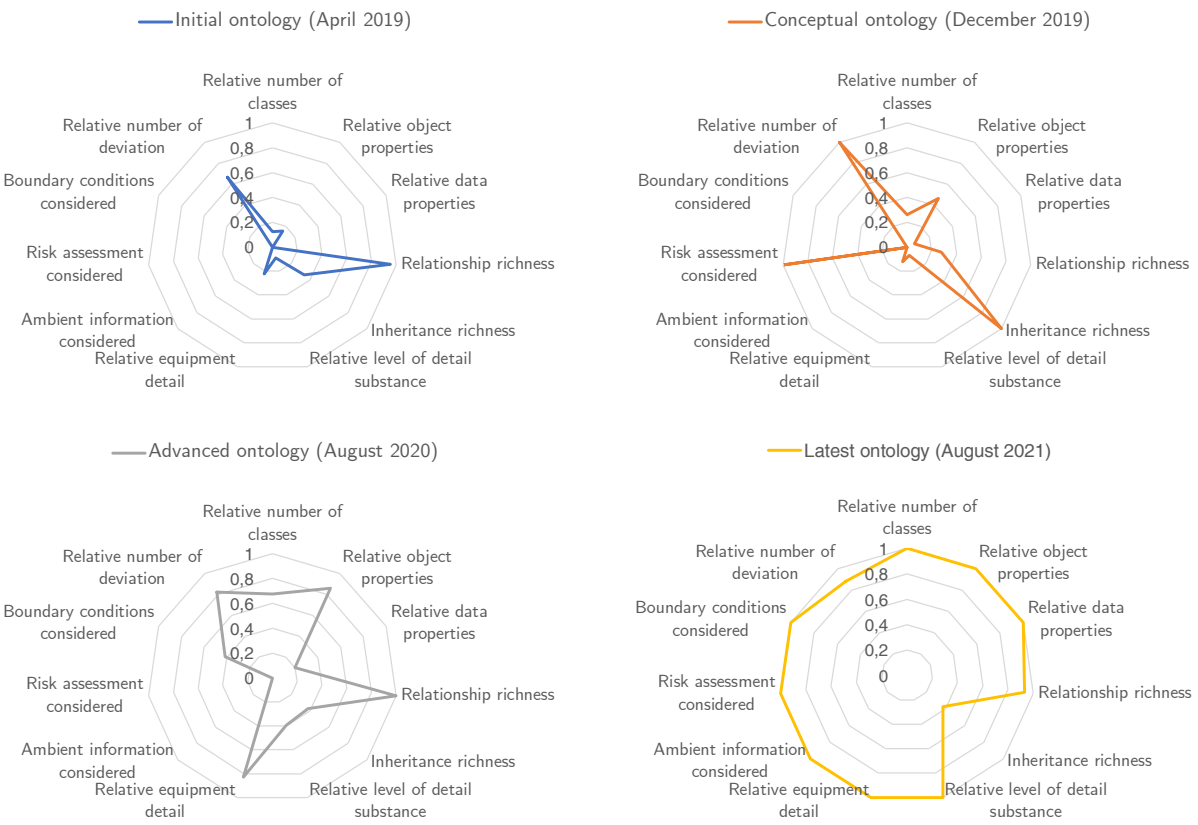


Figure 59: Stages in the ontology development process

5.10.1 Covering typical equipment types

Figure 60 shows the distribution and the absolute number of modeled scenario components concerning the equipment categories considered (cf. Section 5.1). The size of the tiles in Figure 60 is proportional to the number of modeled scenario components and different equipment types. Many scenario components within the ontology have been modeled generically and cannot be assigned to a specific equipment type. Hence, generic concepts are not assigned to a specific equipment entity but are modeled based on structural or functional properties or events within the event sequence of scenarios. Figure 60 shows that the largest number of concepts was modeled generically. Overall, the number of causes modeled is more significant than the number of effects and consequences because they differ the most across scenarios, while consequences differ the least. The number of concepts related to storage tanks is comparable to material transfer equipment, such as pumps and compressors. A comparable number of concepts is modeled regarding process vessel, piping system, and heat transfer equipment. Overall, Figure 60 provides an overview of the type of equipment the focus was placed on within this work.

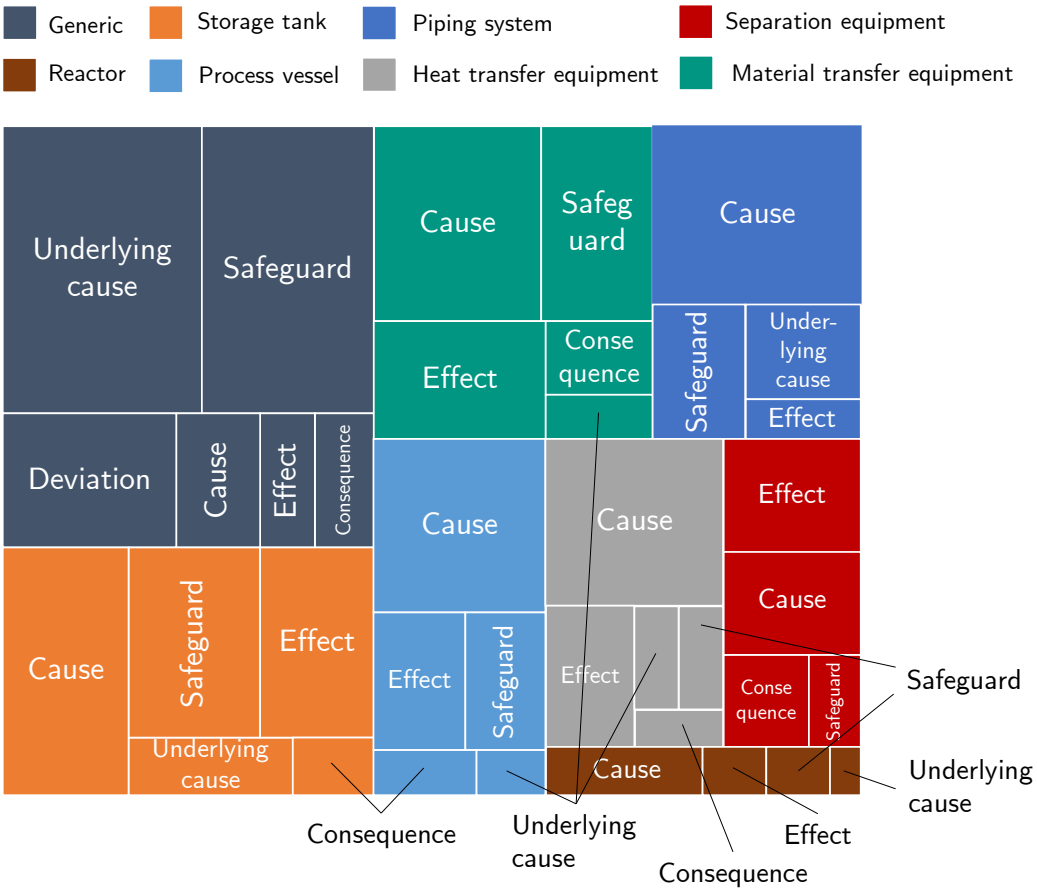


Figure 60: Extent of the modeled scenario components

5.11 Ontology learning

The previous sections show that a curated approach in which class definitions are created manually based on expert knowledge is complex and time-intensive. Gruber (1993) notes that knowledge-based systems involving ontologies are expensive to build, test, and maintain. The design and conceptualization of suitable ontologies require expertise in knowledge engineering and the represented domain. Thus, a manual ontology construction is prone to error and can lead to static knowledge structures which are not scalable. A (semi-) automatic creation and population of ontologies can help reduce the effort to build and maintain them (cf. Single et al., 2020b). Maedche and Staab (2001) established the term *ontology-learning* to describe a mechanism where ontologies are constructed semi-automatically. This research investigated possibilities based on chemical accident databases to automatically populate a knowledge graph based on a manually generated ontology to investigate the possibilities of ontology learning (cf. Single et al., 2020b). Natural language processing (NLP) methods extracted hazardous events, causes, consequences, substance, and location from categorized text to populate a knowledge graph based on these concepts. The ontological model developed is shown in Figure 61. By transforming the accident database into an ontological model the components of the accident scenarios are represented considering the semantic context. Hence, it is possible to search for specific information or accidents using ontology queries and analyze similarities between accident events without searching and analyzing hundreds of database entries. These investigations concerning ontology learning, natural language processing, and accident databases form a side aspect of this research.

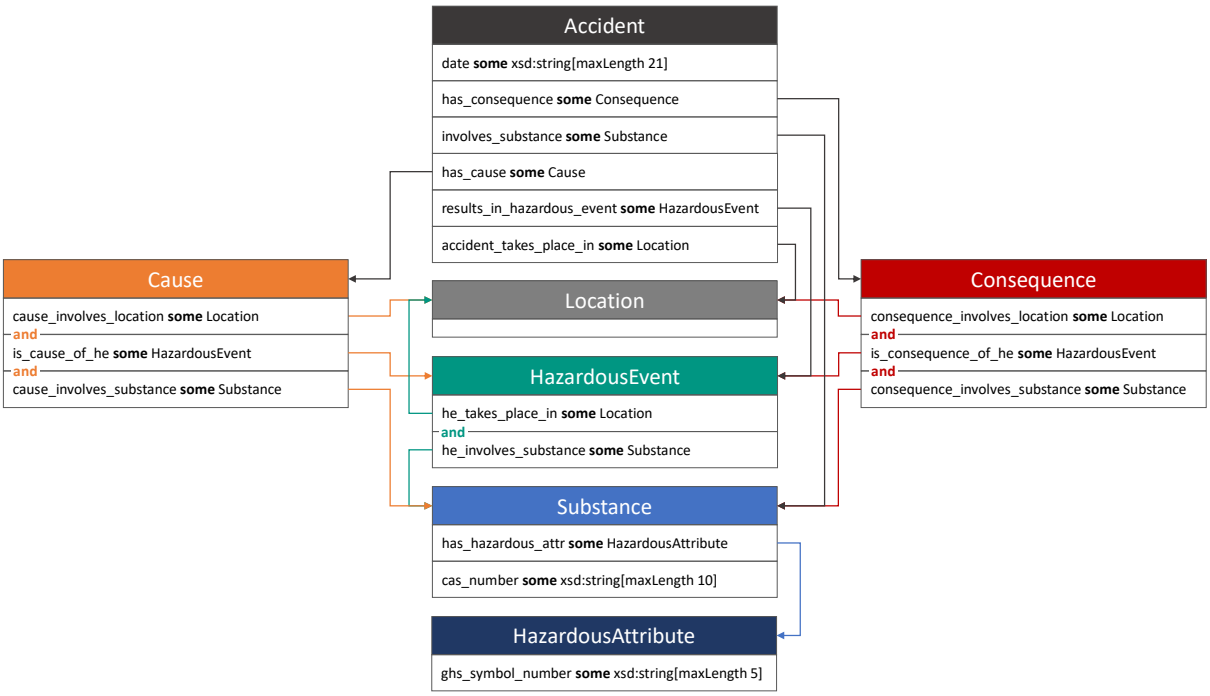


Figure 61: Ontological model described in Single et al. (2020b)

The chemical accident ontology (CAO) developed in this research project is much simpler in structure than the ontology proposed in this dissertation. However, the number of specific scenarios is significantly higher. The different scopes, usage, and type of development (semi-automatically versus curated) can explain this. The scenarios are modeled in their entirety due to the deliberately limited complexity of the scenarios. Static scenario modeling differs from dynamic scenario modeling in this dissertation. Besides, there is no detailed equipment characterization since only the type is distinguished. This simplification made it possible to generate the ontology semi-automatically. The research provided valuable insights into ontology engineering, how to populate semi-automatic ontologies, what possibilities NLP offers, and where its limitations lie.

5.12 Equipment-based scenario identification

After conceptualizing and formalizing an ontology, the contained knowledge must be made available to a software component that exploits it to emulate the HAZOP analysis. Therefore, a reasoning algorithm as an integral part of a knowledge-based system must be designed and developed. Because the ontology is formalized with OWL 2, the HermiT reasoner is available within the Python package Owlready2 (cf. Lamy 2017). The performance of the HermiT reasoner has already been described in Section 3.9.5. Within the inference strategy, logical conclusions are drawn based on the ontology and an input data object in a particular order to generate HAZOP worksheets automatically. The order in which the information is processed and the ontology is evaluated plays a crucial role in efficiently identifying scenarios, assessing risks, and defining suitable safeguards. As described in Section 5.2.1, an evolutionary development approach was chosen for developing, testing, and refining small features successively. Because the ontology cannot be evaluated without an inference algorithm, which changes as the ontology evolves, the requirements constantly change during development.

5.12.1 Inference strategy: equipment-based evaluation

A strategy for evaluating the ontologies is needed to generate HAZOP-like results in worksheets. A distinction is made between an equipment-based and a propagation-based mode to

investigate plant sections with different scopes. The equipment-based mode identifies equipment-specific hazards, equipment failures, and operability problems without considering adjacent equipment. The developed strategy is based on the idea that the desired HAZOP concepts, such as underlying causes, causes, effects, consequences, and safeguards, can be inferred sequentially within an inference cascade. The program flow chart of the inference algorithm is shown in Figure 62. The substance and equipment entity objects serve as an input. Subsequently, all substances involved are processed. The evaluation is conducted separately for each substance category involved. For example, the evaluation for the process medium, e.g., product mixture, and the utility medium, e.g., process steam, are considered subsequently.

Based on the input objects and the selected process deviations, the causes are identified. For this purpose, an instance of the **Cause** class is created. An example instance of the cause class is shown in Listing 17.

```
cause = Cause(isCauseOfDeviation=deviation,
              causeInvolvesEquipmentEntity=[equipment_entity.onto_object],
              causeInvolvesSubstance=[substance.onto_object],
              causeInvolvesSiteInformation=[site.onto_object],
              causeRequiresBoundaryCondition=[equipment_entity.boundary_condition])
```

Listing 17: Exemplary definition of a cause class instance

It contains all cause-related relationships (properties) and all required input instances. Afterward, the HermiT reasoner is called, which performs the classification and identifies the cause classes, which fulfill the restrictions of the input data. Based on the identified causes, underlying causes and effects are inferred. Within this model representation, underlying causes always depend on causes, while effects depend optionally on causes. Thus, the reasoner can simultaneously identify underlying causes and effects since causes are the input for both concepts. The sequence in which the scenario components are identified does not necessarily have to be the same as the sequence of the scenarios (cf. Section 0). After identifying effects, subsequent deviations initiated by effects are inferred using case-based reasoning (CBR). This step is also referred to as *backward reasoning* and is described in more detail in Section 5.13. If subsequent deviations are identified, these are provided with an explanation because the deviations are based on effects and are evaluated in a separate routine. Afterward, an attempt is made to identify consequences based on previous findings. The entire consequence sequences are considered (cf. Section 5.12.2). After consequences have been identified, the safeguards that prevent or mitigate effects and consequences are identified. Finally, the results are processed for documentation in the form of extended HAZOP worksheets.

The Python code of the implementation of the equipment-based mode is shown in Appendix Listing Q-1, Appendix Listing Q-2, and Appendix Listing Q-3. In summary, the equipment-based inference algorithm can be divided into five parts:

1. Causes are identified based on a deviation, substance, mode of operation, intended function, and the equipment entity object, including all required boundary conditions and site information. The identification of causes forms the basis for further inferences.
2. The effect, underlying cause, and consequence instances are created and classified by the reasoner depending on each other (cf. Figure 31). This block was combined to minimize the number of reasoner calls. If no scenario components are found for specific combinations of deviations, substances, mode of operations, intended functions, and equipment entities, the algorithm continues without performing the following steps.
3. Subsequent deviations are inferred based on effects and causes using case-based reasoning. Besides, subsequent deviations based on other deviations are identified (cf. Section 5.13.1).

4. Afterward, subsequent consequences are inferred based on the primary consequences. In addition, the risk assessment of scenarios is performed by identifying likelihood and severity based on the preliminary results (cf. Section 5.8.4). Additionally, cause-related preventive safeguards, and consequence-related mitigative safeguards, are identified.
5. Finally, the findings are processed, and a HAZOP worksheet is generated.

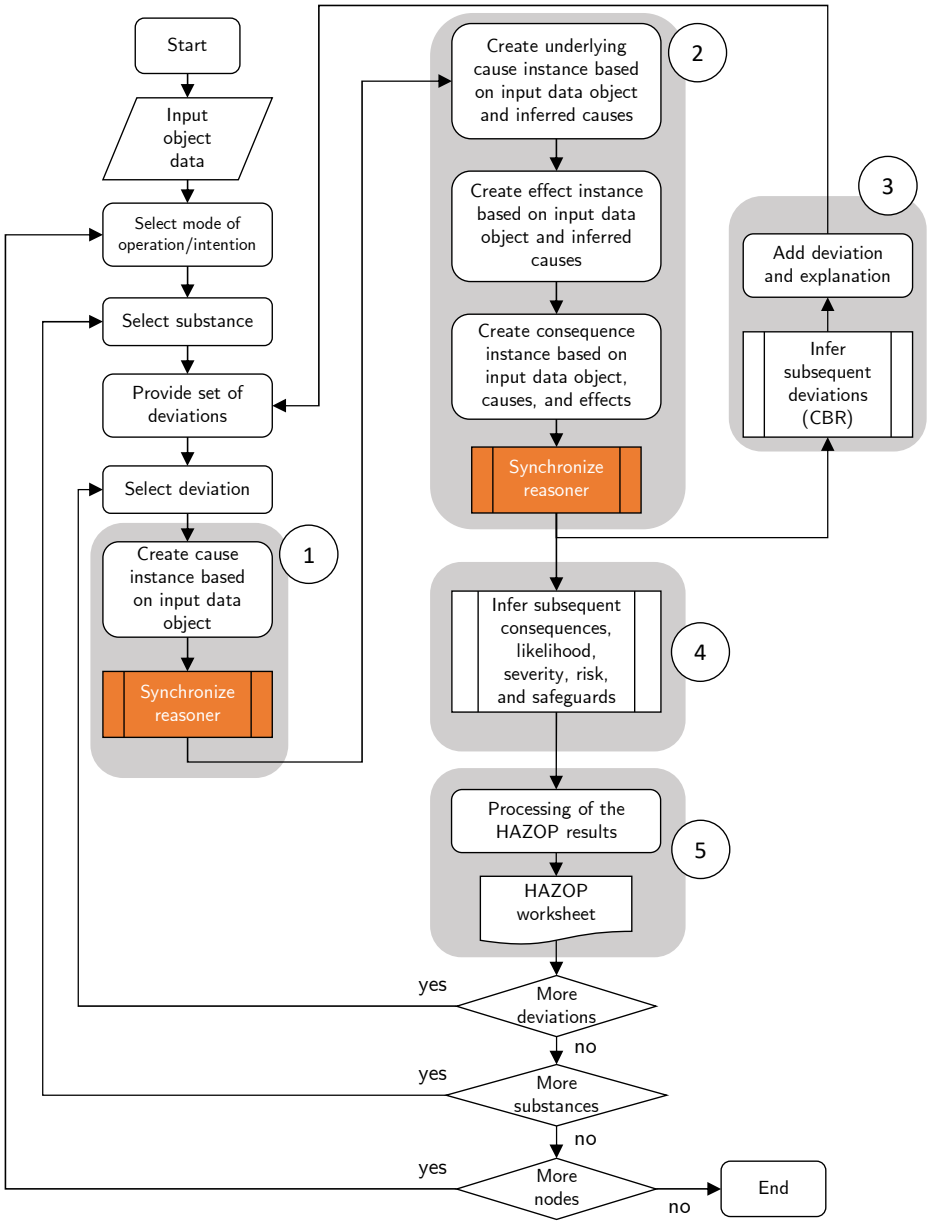


Figure 62: Inference algorithm: equipment-based mode

5.12.2 Inference strategy: consequence sequence

The sub-function for identifying consequences and consequence sequences is described in the following. A flow chart of the sub-function for identifying consequence sequence and safeguards is shown in Figure 63. The function can be organized into three parts:

0. Preparation: the input data (deviation, substance, mode of operation, intended function, equipment entity object, site information, preliminary results) is provided to infer subsequent deviations.
1. Subsequent consequences are identified based on the input data. These are derived primarily based on the primary consequences.

2. Instances are generated to identify the cause-related likelihood, the consequence-related severity, the resulting risk of the scenario, and appropriate safeguards. By calling the reasoner, the instances are classified, and conclusions are drawn.

Finally, all results are arranged to represent different consequence sequences and possible parallel scenarios. Examples of consequence sequences are

- "PumpBreakdown" → "ProductionDowntime",
- "LossOfPrimaryContainment" → "FireHazard",
- "LossOfPrimaryContainment" → "RiskOfExplosiveAtmosphere".

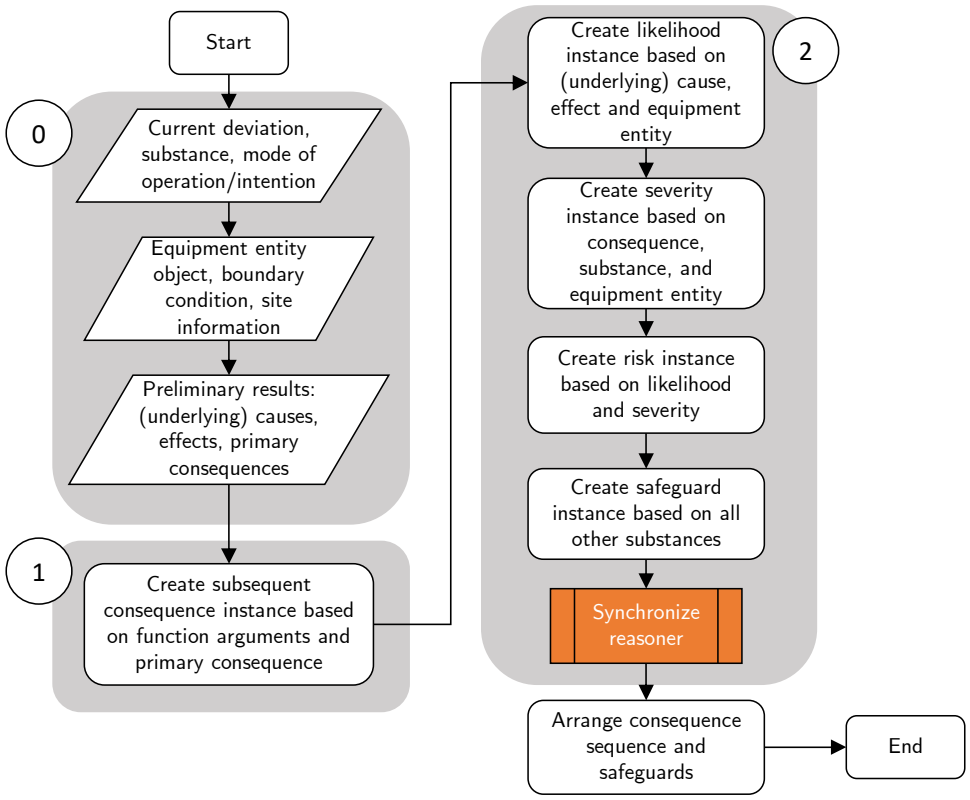


Figure 63: Sub function: consequence cascade and safeguards

5.13 Identification of subsequent deviations

Figure 64 shows references within a typical HAZOP worksheet, such as the 'high level' (1.1) deviation has the potential consequence of 'high pressure' (cell 1.1.1.1). The deviation 'high level' that leads to the consequence 'high pressure' is listed in the next row as the cause (cell 1.2.1) of the deviation (cf. Figure 64). This circumstance is described by Duhon (2017) as 'guideword overlap', e.g., when flow deviations lead to other deviations. It is impossible to use a **Deviation** as a **Consequence** within the knowledge-based system since both classes are disjoint within the ontology, which means they cannot share any instance (cf. Section 5.9.1). The need arises from these illustrated relationships to represent the conversion of deviations into other deviations. Consequences or effects that implicitly cause a deviation must be recognized and considered. Hence, an efficient reasoning mechanism is needed to draw appropriate conclusions. Besides the forward reasoning based on (semantic) reasoners (cf. Section 5.12.1), a mechanism for backward reasoning based on case-based reasoning (CBR) was developed to enhance the reasoning capabilities.

Node description	Hexane storage tank with inflow line
Drawings / References	P&ID: D-007;
Design Intent/Parameters	Storage of liquid hexane, MAWP = 0.5 barg @ 293 °C

Deviation	Causes	Cause Likelihood	Consequences	Consequence Severity	Safeguards	Unmitigated Risk Rankings	Mitigated Likelihood	Mitigated Risk Rankings	Recommend ations
1.1 High level	1.1.1 Flow not discontinued	3	1.1.1.1 High pressure (see 1.2)	2	1.1.1.1.1 Level indication with high level alarm	2			
1.2 High pressure	1.2.1 High level (see 1.1)	3	1.2.1.1 Loss of containment	4		3			
	1.2.2 High temperature	3	1.2.2.1 Exceeding MAWP, collapse of enclosure	2		2			

Figure 64: References between scenarios within a typical HAZOP worksheet

The idea behind forward and backward reasoning is shown in Figure 65. Backward reasoning is used to identify subsequent deviations that may result from previous deviations, causes, or effects (cf. Figure 65). Using backward reasoning makes it possible to represent more complex relationships within scenarios than the causal chains modeled within the ontology. Thus, instead of inferring causes and effects based on the deviation (forward reasoning), further deviations are inferred from deviations, causes, and effects (backward reasoning) (cf. Figure 65). The Python code for determining subsequent deviations is shown in Appendix Listing Q-4. For example, the deviation **HighTemperature** within a tank is the cause of the process deviation **HighPressure** (cf. Figure 64, cell 1.2.2). Thus, **HighPressure** is a subsequent deviation of **HighTemperature**, which can be identified using backward reasoning. The extended possibilities resulting from backward reasoning are illustrated by a scenario shown in Figure 66. An operational error leads to a cooling failure within a reactor resulting in an increased temperature.

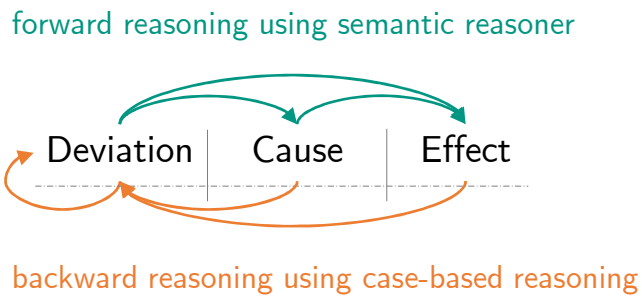


Figure 65: Forward and backward reasoning

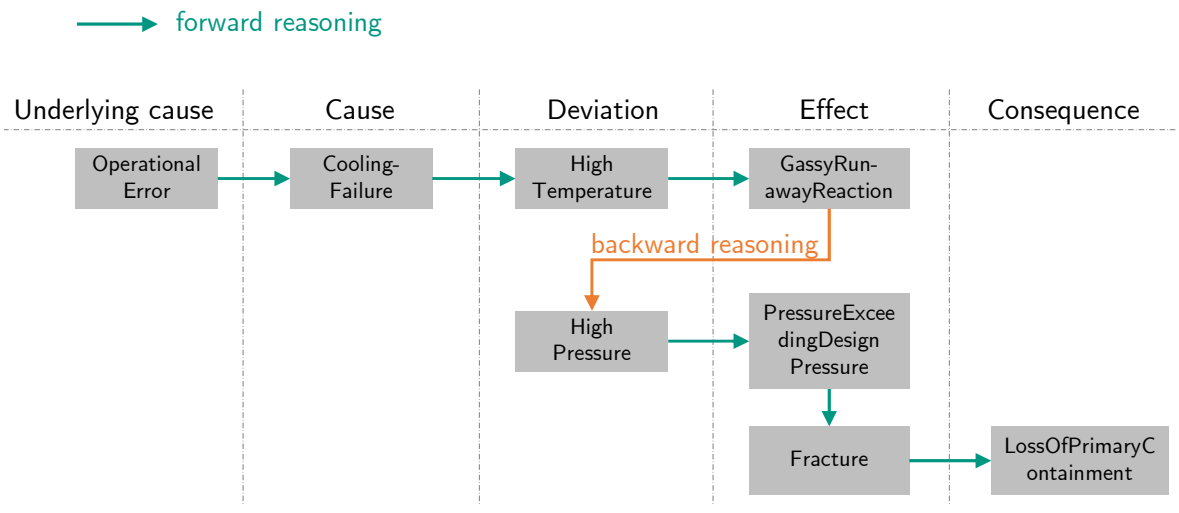


Figure 66: Partial cause-effect chain concerning a runaway reaction

Due to the increased temperature, a gas-forming runaway reaction is triggered, which increases the pressure, possibly exceeding the vessel's design pressure (reactor) with a subsequent fracture, which yields a loss of primary containment. Up to the effect of "GassyRunawayReaction", the described scenario can be determined with the help of forward reasoning. The resulting **HighPressure** cannot be concluded with a (semantic) reasoner because effect and deviation are semantically disjoint. Hence, the resulting deviation **HighPressure** is identified using backward reasoning based on CBR. Additionally, an explanation is added to describe the partial scenario leading to the **HighPressure**. This step is shown in Figure 62 in section ③. Backward reasoning considers that the effect of one scenario is the cause (or chain of causes) of another scenario. Thus, forward and backward reasoning strategies are used in conjunction. The backward reasoning step often explains the subsequent scenario. Without backward reasoning, relationships, as shown in the example in Figure 66, could not be represented because the causal sequence within the ontological model specifies that an effect results from a deviation and not vice versa. The explanation describing the causal sequence leading to the deviation is shown in the cause column in the HAZOP worksheet to explain the backward reasoning step (cf. Section 6.1.2). The explanation includes the cause, the effect, and the resulting deviation. For example, explanations for backward reasoning look like this:

- T400: ContaminationByWater → BacteriaGrowth → HighCorrosion (Multiphase),
- T400: PumpIncorrectlySet → VolumetricDisplacement → HighPressure (Multiphase).

5.13.1 Case-based reasoning for backward reasoning

Identifying subsequent deviations from effects and former deviations described in Section 5.13 requires a representation of cases for modeling these coherences. In this work, a method was developed to identify subsequent deviations using case-based reasoning (CBR) (cf. Section 3.9.3, Figure 65). The use of CBR to identify subsequent deviations has been described by Single *et al.*, 2020e. The basic idea of CBR is to use information from past situations (cases) to conclude new situations (cases). The reasoning method is based on a case base (cf. Table 18), including 189 cases (cf. Appendix E). The entire case base was created manually within various case studies and adapted to the requirements of the method development process. For example, an increased temperature in a gas-filled vessel can increase pressure. Therefore, the subsequent deviation 'high pressure' can be characterized by the deviation 'high temperature', the pressure vessel, and the state of aggregation. Besides deviations, effects inferred by the inference algorithm can also imply subsequent deviations, e.g., abnormal evaporation within a storage tank can lead to increased pressure. Each of these cases is characterized by attributes and integrated into the case base. The number of attributes must be kept minimal to limit the modeling effort, but cases must also be distinguishable. Information regarding the trigger event, the apparatus in which the event occurs, and the state of aggregation of the substance involved are needed to identify subsequent deviations.

In a preliminary stage of development, attributes such as the number of ports of the equipment and adjacent equipment were considered. They did not add value for delimiting cases and were discarded later (cf. Single et al., 2020d). Besides, the intended functions of the equipment entities characterize the cases to consider functional aspects. Within this research, the following five case attributes a_i have been identified for sufficiently characterizing cases (cf. Table 16):

- *apparatus*: one of the essential attributes because it describes the structural aspect of the enclosure,
- *intended function*: an attribute that contributes information about the process and functional aspects,
- *substance phase*: attribute providing the essential substance property since deviations can behave differently in a different state of aggregation,

- *trigger event*: deviation, cause, or effect; most crucial attribute, because it describes the trigger of a subsequent deviation,
- *equipment entity name*: attribute which provides additional information in case of structural similarity.

In addition to the attributes, each case contains the prediction (inference) of the subsequent deviation. Several subsequent deviations may occur. An example case is shown in Table 17, Case 5, where a **LowFlow** into a **PressureVessel** leads to a **LowLevel** deviation. The equipment entity name attribute is not defined because this case applies to all equipment entities that involve the apparatus pressure vessel. This example does not define the intended function attribute because multiple intended functions are conceivable without invalidating the case.

During the reasoning process, similarity measures between the case being evaluated and the cases in the case base are determined to identify the best match by comparing these similarity measures. In the inference algorithm, this step is shown in Figure 62 in section ③. The similarity measure depends strongly on the data type of the case attributes (cf. Section 3.9.3). Since the attributes correspond to the data type string, a binary comparison of attributes was chosen. Accordingly, there are only binary states when comparing case attributes; either the attributes are identical or not. Thus, the local similarity measure s of case attribute a_{Ai} of case A and attribute a_{Bi} of case B is determined as follows:

$$s(a_{Ai}, a_{Bi}) = \begin{cases} 0 & if\ a_{Ai} \neq a_{Bi} \\ 1 & if\ a_{Ai} = a_{Bi} \end{cases}. \tag{11}$$

Local in this context means that the similarity measure refers to one attribute and not all five attributes that describe the entire case. Since numerical values are not included in the attributes of the cases a_i (cf. Table 16), a binary consideration is sufficient when determining the local similarity measure from this perspective. Alternatively, the similarity measure between two string objects could be determined using the Jaro-Winkler distance (cf. Cahyono, 2019). The determination of the distance metrics results in values between [0...1]. For instance, the Jaro-Winkler distance between the strings "CompressorEntity" and "ReactorEntity" is 0.696.

Despite the fundamentally different meaning of the strings, there is a relatively high degree of similarity. Table 16 shows other examples of Jaro-Winkler distances between case attributes. The comparison of the names of the equipment entities results in a Jaro-Winkler distance of 0.603. The structurally different apparatuses "AtmosphericStorageTank" and "PressureVessel" result in a distance measure of 0.406. The process-related fundamentally different intended functions "Storage" and "Evaporation" still result in a distance measure of 0.40. Since the *substance phase* attribute is identical, the distance measure is 1.0. For the opposing effects "AbnormalEvaporation" and "IncompleteEvaporation", the result is 0.768.

Table 16: Jaro-Winkler distance of different case attributes

Case Attribute a_i	**Candidate 1**	**Candidate 2**	**Jaro-Winkler distance**
equipment entity name	StorageTankEntity	ShellTubeEvaporatorEntity	0.603
apparatus	AtmosphericStorageTank	PressureVessel	0.406
intended function	Storage	Evaporation	0.400
substance phase	Multiphase	Multiphase	1.0
initial event	AbnormalEvaporation	IncompleteEvaporation	0.768

The distance measure of 0.768 is clearly too high for two opposite effects and leads to wrong conclusions. From this, it can be concluded that, concerning the attributes of the cases, the determination of the similarity measure not based exclusively on binary comparisons does not lead to better results. A purely formal comparison of the structure of the strings of the attributes is insufficient to determine the similarity of the underlying concepts. Other distance measures such as Levenshtein (cf. Hofmann, 2018) do not provide better results, and the binary comparison is sufficient for this work.

The total similarity measure S_T between Case A and Case B is determined based on the local similarity measure s_i and the weight w_i as follows:

$$S_T(A,B) = \sum_{i=0}^{n=5} s_i(a_{Ai}, a_{Bi}) \cdot w_i \tag{12}$$

A similarity measure of 1 would mean a perfect match between two cases. The case with the highest similarity measure is accepted during the reasoning process. The higher the weight of an attribute w_i, the more important is the attribute. Furthermore, the following applies $\sum w_i = 1$. By changing the weights of the attributes, the relevance of these attributes is changed. The weights are assigned globally over the complete set of cases within the case base. The Python functions for calculating the similarity measure are shown in Appendix Listing E-14. The usage of uniform weights w_i for all five cases would mean that all attributes a_i are equally important. For instance, the attribute equipment entity name would be as important as the apparatus, which is not plausible since the apparatus uniquely characterizes the structural enclosure. Besides, the name provides only additional information (cf. weights in Table 17). Using the same weights can lead to redundancies or skewness when calculating similarity measures leading to wrong conclusions. Especially with inhomogeneous datasets, non-uniform weights have the advantage that dominating attributes are still considered, which simplifies the transferability of cases. Additionally, a threshold regarding the similarity measure is considered before a case is accepted as similar enough (cf. Figure 67). Without the threshold, the case with the highest similarity measure would be accepted, even if it is exceedingly small and the cases show little similarity. For instance, a maximum similarity measure of 0.1 would be accepted, indicating barely any similarity. This threshold also depends on the assigned attribute weight.

Different weight constellations were examined, evaluated, and adapted to increase the reasoning performance. The reasoning performance refers to the correctness of the conclusions drawn. Figure 67 shows how the weights of the attributes influence the threshold, and the correctly recognized cases are compared. Therefore, all attributes are weighted equally in one set, which means there are no weights (circular symbols in Figure 67). In the other set, the weights from Table 17 are selected (rectangular symbols in Figure 67). In determining the attribute weights, 16 case samples were created, which are not represented in the case base, which serve as test data. The case samples are shown in Appendix V. These case samples can be used to evaluate the performance of CBR in the context of the assigned weights of the attributes. The 16 case samples are unknown (new) cases for which similarities to the cases in the case base are to be identified. The sample cases are not contained in the case base because this would directly result in a similarity measure of 1. The performance of CBR is evaluated individually for each case sample which is shown in Figure 67. All correctly recognized cases are marked with filled symbols in Figure 67. If the attributes are weighted, all cases above a threshold value of 0.75 are recognized correctly. Accordingly, there are no unfilled squares above this threshold in Figure 67. If this threshold is not reached, it is assumed that no existing case is similar enough to be transferred to the current case. This differentiation is no longer easily possible if no specific weights are used (circle symbols in Figure 67).

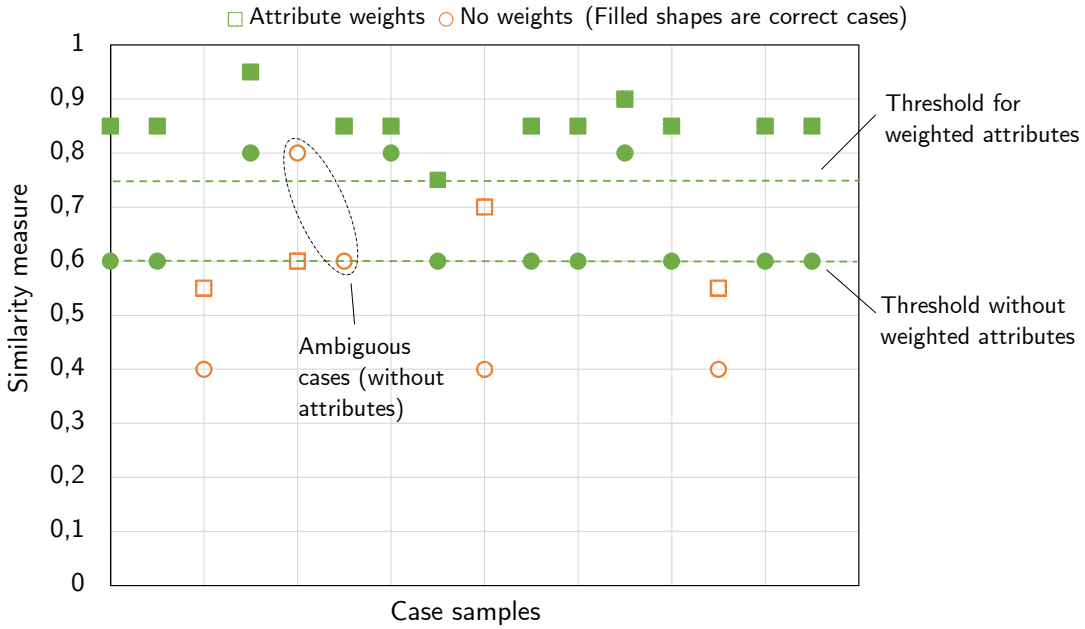


Figure 67: Influence of attribute weighting on the threshold

In Figure 67, a clear case distinction is no longer possible for two cases, which leads to ambiguous cases. It follows that with a similarity measure of 0.6 or above, both correct and wrong conclusions are possible, and thus an unambiguous threshold can no longer be defined. The evaluation of these case samples shows the importance of carefully chosen attribute weights and the associated thresholds. Thus, weighting the attributes helps to consider dominant features in particular to draw the correct conclusions. Table 17 shows examples of global similarity measures between different cases. The similarity measure between case 1 and case 2 is 0.6 despite four identical attributes since the most crucial attribute *trigger event* differs. In case 3 and case 4, the cases from Table 16 are compared.

Table 17: Example cases for identifying subsequent deviations

Attribute a_i	**Weight w_i**	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Case 6**
equipment entity	0.05	PumpEntity	PumpEntity	StorageTank-Entity	ShellTube-Evapora-torEntity	None	Stabi-lizerCol-umnEntity
apparatus	0.3	Casing	Casing	Atmospheric-StorageTank	Pres-sureVessel	Pres-sureVessel	Pres-sureVessel
intended function	0.1	Material-Transfer	Material-Transfer	Storage	Evapora-tion	None	Stabiliza-tion
substance phase	0.15	Liquid	Liquid	Multiphase	Multiphase	Multiphase	Multiphase
trigger event	0.4	Cavitation	PumpRun-ningDry	Abnorma-lEvaporation	Incomplete-Evapora-tion	LowFlow	LowFlow
Similarity measure S_T		0.6		0.15		0.85	
Inferred deviation		High-Vibration	HighTem-perature	HighPressure	Other-ThanCom-position	LowLevel	LowLevel
		-	High-Vibration	-	-	-	-

Since the trigger event and the apparatus, which are dominant attributes, differ, the similarity measure is 0.15. Ergo, the fundamental differences between cases 3 and 4 are correctly recognized. Therefore, these cases would not be accepted due to the weight-dependent threshold of 0.75. Case 5 shows a low flow deviation resulting in a low level within a pressure vessel. The unspecified attributes *intended function* and *equipment entity name* result in a generalized case that can be transferred easily. Due to the correspondence of the most significant attributes, a similarity measure of 0.85 between case 5 and case 6 is above the threshold of 0.75 and would be accepted.

5.13.2 The rationale for case-based reasoning

The following requirements serve as criteria for selecting a suitable reasoning paradigm for backward reasoning: short runtime and conclusions should be drawn even for incomplete cases. A CBR paradigm is used for backward reasoning within this research work. Single *et al.* (2020c) described the use of support vector machine (SVM) classification to identify subsequent deviations. Further research within the scope of this dissertation showed that CBR is more suitable for this task. Since specific cases only occur once in the case base (cf. Appendix E) and each case is described by the five attributes (features), the SVM model was insufficiently trained for individual cases, leading to classification problems, which worsens the overall reasoning performance. Besides, several subsequent deviations (cf. case 2 in Table 18) lead to a multi-class classification that exacerbates the mentioned training issues. Because of the manual creation of SVM training cases based on domain knowledge (cf. Appendix E), the potential advantages of SVM classification based on aggregated real-world data do not come into effect. Due to the lack of advantages of SVM compared to CBR, SVM is not further used in this work to identify subsequent deviations.

Alternatively, the ontology could be extended by further relationships between the core concepts to perform backward reasoning utilizing a (semantic) reasoner. Backward reasoning would require the introduction of inverse properties between **Effect**, **Cause**, and **Deviation** classes and additional restriction on the **Deviation** classes. Figure 68 shows the inverse property deviationCausedByEffect between the core concepts **Deviation** and **Effect**. The corresponding example is based on case 2 from Table 17. The effect "PumpRunningDry" is identified based on the NoFlow deviation in the forward reasoning step. In the backward reasoning step, the subsequent deviations of the effect "PumpRunningDry" **HighTemperature**, and **HighVibration** are identified using the HermiT reasoner. Within this work, the fact could be observed and confirmed that the introduction of inverse properties within the developed ontology negatively affects the overall reasoning performance. Other researchers previously described this phenomenon. Liebig (2006) noted that inverse properties disable efficient caching methods within tableaux-based DL reasoners.

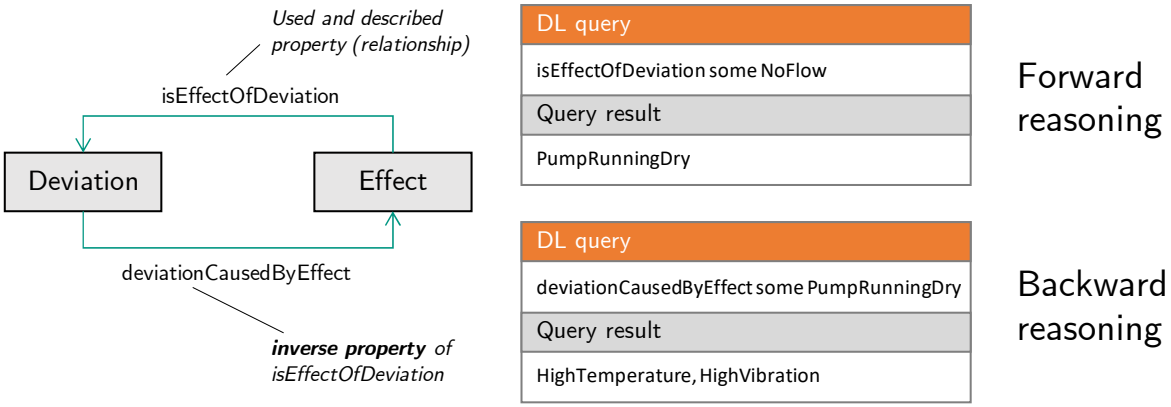


Figure 68: Example of an inverse property including queries and query answers

Since the used HermiT reasoner is a tableaux-based DL reasoner, this issue also applies here. After the reasoner is called within the reasoning algorithm, it takes a particular time where facts are asserted, and the reasoning is conducted. The runtime of the HermiT reasoner call is compared to the runtime of a CBR evaluation. The reasoner runtime depends directly on the structure, size, the number of individuals in the ontology, and the computing power. Two machines were compared, machine A with an Intel Core i7-5600U CPU@ 2.60 GHz, 16 GB RAM, and machine B with an AMD Ryzen 5 2600 CPU@ 3.40 GHz, and 16 GB RAM. For example, the typical average runtime of the HermiT reasoner within the Python package Owlready2 without inverse properties during forward reasoning was:

- Machine A, an average of 10.39 seconds per call (155 reasoner calls),
- Machine B, an average of 5.82 seconds per call (155 reasoner calls).

By adding inverse properties (cf. Figure 68) and the necessary restrictions, the runtime of the reasoner was significantly extended:

- Machine A, an average of 28.48 seconds per call (155 reasoner calls),
- Machine B, an average of 10.29 seconds per call (155 reasoner calls).

The runtimes are shown in Appendix F. Depending on the size of the process plant, number of operation modes/intended functions, number of deviations, and substances, several hundred reasoner calls may be needed to evaluate the process section automatically. Hence, calling the reasoner is one of the most computationally expensive parts of the algorithm. The computer code has been optimized to reduce the number of reasoner calls. In this course, the scenario components **UnderlyingCause**, **Effect**, and **Consequence** were evaluated simultaneously with one reasoner call instead of executing a separate reasoner call for each scenario component (cf. Section ② in Figure 63). The consequence sequences, risk assessment, and identification of safeguards are also performed with one reasoner call per deviation, equipment entity, and mode of operation/intended function (cf. Figure 63). Furthermore, in the inference algorithm, care was taken to perform operations simultaneously instead of sequentially whenever possible.

The average run time of one CBR call is 0.005 seconds for machine A and 0.0018 seconds for machine B. Regarding the runtimes, it should be noted that with CBR, only numerical similarity measures between cases are calculated, while when running the reasoner, all facts within the ontology are asserted. Another reason CBR is used backward reasoning is that no exact model is required, and conclusions can be drawn based on incomplete information (cf. Table 7). Hence, valid inferences can be drawn without specifying every attribute of the cases (cf. case 5 in Table 17). Besides, reasoning becomes more efficient with new cases. New cases can be added without maintaining old cases since the similarity measure for the new cases more precise cases will be higher automatically. Within ontologies, implicit relationships between concepts are represented via semantic relationships (properties) and axioms. CBR does not represent implicit relationships between concepts. Due to the different reasoning principles, combining semantic reasoning with case-based reasoning is reasonable to take advantage of both paradigms. Therefore, within this work, both reasoning paradigms are used together.

5.14 Propagation of hazards and operability issues

A purely equipment-based scenario identification without considering the interactions between equipment entities can lead to scenarios not being recognized. The causes of process deviations can be found upstream of the equipment under consideration, while effects can occur in downstream equipment, leading to subsequent deviations. In addition, reverse flows or unacceptable pressure buildup in downstream equipment can be the causes of scenarios in upstream equipment. A propagation routine was developed to solve this issue. The propagation mode of the inference

algorithm contributes significantly to ensuring the plausibility of scenarios by providing causes. The propagation mode potentially provides more profound results for all types of plant section topologies. Following the equipment-based analysis, where the entire set of deviations is applied to all equipment entities, the propagation-based analysis is performed.

Figure 69 shows the interaction between the equipment-based and the propagation-based mode. The SDG-based process plant representation and the node-specific substance, equipment entities, boundary conditions, and environmental information are the input for the algorithm. During the equipment-based analysis, *partial scenarios* (cf. Figure 28) that qualify for propagation with no immediate effect or consequence are stored within equipment-specific lists (cf. Figure 69). These partial scenarios within these lists typically consist of a deviation, underlying cause, cause, and effect. This process is outlined in detail in Section 5.14.3. Within the first step of the propagation-based analysis, the topology of the process plant section is analyzed (cf. Figure 70), and the nodes are rearranged (cf. Section 5.14.2). Subsequently, all nodes of the rearranged plant sections are traversed. In the context of the investigation of the propagation of events, it was found that partial scenarios consisting of an underlying cause, cause, and deviation without immediate effects or consequences can arise in equipment entities (cf. Figure 28). The propagation of partial scenarios through equipment entities may result in complete scenarios in adjacent equipment entities. During the propagation process, equipment-specific lists of partial scenarios are used for storing and propagating partial scenarios. If the evaluation of partial scenarios results in subsequent deviations or has no immediate effects within the current equipment, these findings are propagated further through the plant section. After complete scenarios are identified, a risk assessment is performed. This procedure is repeated for all nodes within the current line under consideration and continued for every line of the entire process plant section. The Python code of the implementation is shown in Appendix Listing Q-5, Appendix Listing Q-6, Appendix Listing Q-7.

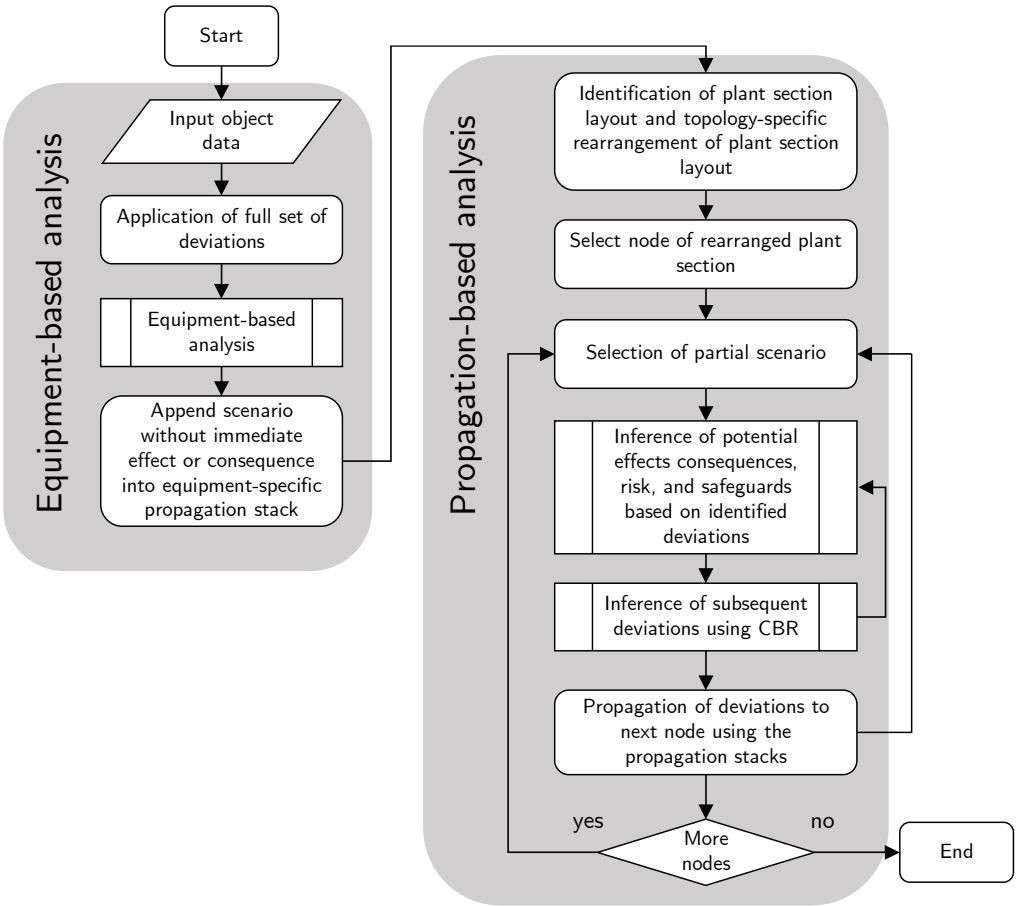


Figure 69: Overview propagation and equipment-based analysis

5.14.1 Inference strategy: process plant section topology

The propagation strategy depends on the topology of the plant section. The term plant section topology describes the physical structure of equipment, including its connectivity. Different plant types were analyzed in this research to derive basic plant section topologies from which entire plant sections are composed. The CSE High-Pressure-Loop, an ammonia refrigeration plant, and a stabilization column plant are transformed into an SDG representation to investigate the nodes. The associated P&IDs and SDG representations are shown in Appendix Figure R-1, Appendix Figure R-2, Appendix Figure R-3, Appendix Figure R-4, and Appendix Figure R-5. Figure 70 shows the derived basic plant section topologies that can be used to compose complete plant sections (cf. Figure 71). A distinction is made between single-line, recycle flow, junction, single-cycle, multi-cycle, and branch systems. These represent basic configurations from which more complex topologies can be created. In Figure 70, initial nodes are marked with a circle around the node. A single stream system consists of equipment entities arranged in a line. Within this system, deviations can be propagated from one end to the other. In recycle stream systems, the nodes are traversed to the first node, leading backward (cf. node 3 in Figure 70 b). Then the nodes are traversed downstream to the last node. For Figure 70 b, the strands [1, 2, 3], [3, 4, 2, 3], and [3, 5] would result accordingly. In systems that form a junction, the initial deviations are applied to node one and node two (cf. Figure 70 c). After the merge, the deviations and other propagating events from both lines are propagated further downstream (cf. Section 5.14.2).

Within circular processes, the nodes are passed through one after the other. The system checks whether further deviations or other events were identified after the first run and are considered in a second run (cf. Figure 70 d). The first step within a multi-cycle system (cf. Figure 69 e) is to identify the node where the two circular processes intersect. Then, the initial deviations are propagated from the first node after the intersecting node in the first cycle. After the deviations are propagated through the first circular process, the deviations identified within the intersecting node are propagated through the second circular process. If additional propagating events arise within the intersecting node in the second cycle, they are propagated again through the first cycle. In branched systems (cf. Figure 70 f), the deviations are propagated until the branch and afterward through both node lines. After the branch, both lines are considered separately analogous to the single-stream systems. A purely equipment-based analysis provides insufficient explanations for potential process deviations, especially when multiple lines end in the same equipment entity.

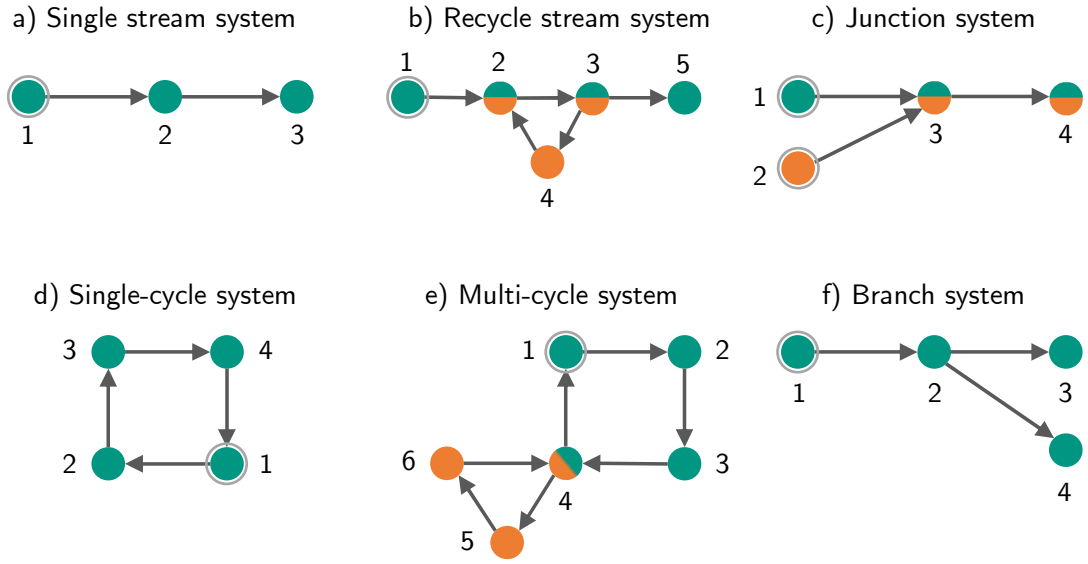


Figure 70: Basic plant section topologies

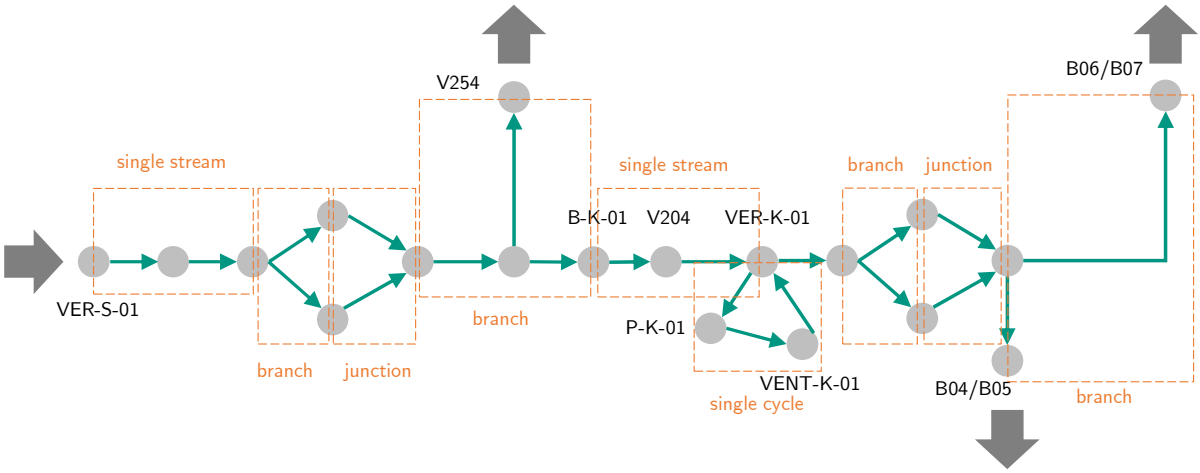


Figure 71: Signed directed graph representation of an excerpt of the CSE High-Pressure-Loop highlighted basic plant section topologies

5.14.2 Algorithm for the identification of plant topologies

The flow chart of the developed algorithm for analyzing process sections is shown in Appendix Figure R-6. It decomposes the topology into smaller sections evaluated separately considering the interfaces. In principle, the plant section layouts shown in Figure 70 can be distinguished, and for the differentiation of the basic plant section topologies, a separate subfunction is used. This subfunction involves tracing paths within the graph, identifying cycles, recycle flows, and calculating inflow/outflow ratios of nodes. The outflow/inflow stream ratio is calculated from the number of streams n_{in} into the node k, and the number of streams n_{out} out of node k as following:

$$r_k = \frac{n_{out,k}}{n_{in,k}}. \tag{13}$$

The outflow/inflow stream ratios are directly derived from each node based on the edges. For example, if the number of outflow and inflow streams is equal, the outflow/inflow stream ratio is 1. For the first node within the SDG, the outflow/inflow stream ratio is ∞ since there is no inflowing stream ($n_{in,k} = 0$) resulting in $r_k = \infty$. Based on the outflow/inflow stream ratios, different plant section layouts can be identified based on the pattern of the ratios. The basic plant section topologies from Figure 70 result in the following patterns:

- A single-stream system has the pattern [∞, 1, ..., 1, 0].
- Recycle stream systems contain a cycle, and the outflow/inflow stream ratios of the inner nodes differ strongly, e.g., [∞, 0.5, 1, 2, 0].
- A single-cycle system has a pattern [1, ..., 1].
- Every node has both an inflow and an outflow within a multi-cycle system, and the ratio pattern is [1, ..., 1]. Additionally, the number of inflows and outflows must be considered to recognize both cycles' intersections because the interconnection ratio is 2/2, and of the other nodes, 1/1.
- A 'branch system' pattern depends on the number of branches, such as [∞, 2, 0, 0] or [∞, 3, 0, 0, 0].

However, a purely inflow/outflow ratio pattern-based analysis is insufficient since fundamentally different systems can also have the same pattern. For example, the topologies in Figure 72 have the same pattern.

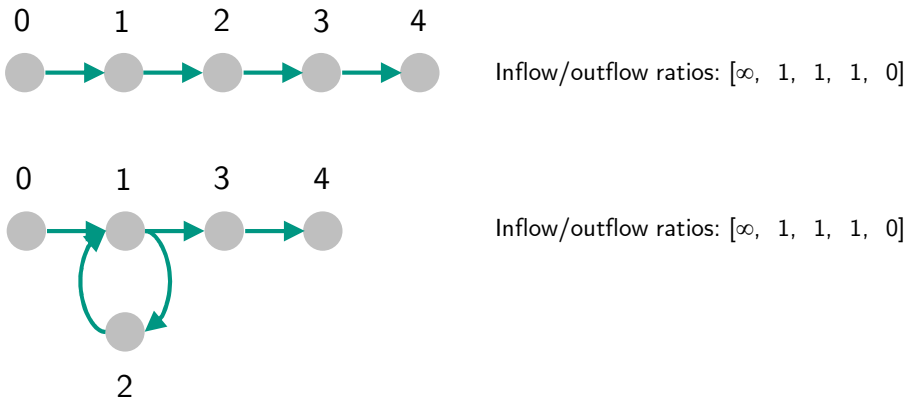


Figure 72: Inflow/outflow ratios of different topologies

Thus, besides the inflow/outflow ratios, a graph is examined for cycles. Figure 73 shows two topologies composed of several basic plant section topologies. In this case, the algorithm attempts to decompose the entire topology into basic topologies, which are further decomposed. The result of such a decomposition is shown in Figure 73. This procedure ensures that deviations and events are propagated through all subsequent nodes and lines throughout the plant section. In case the complexity of the plant section exceeds the presented schematic basic topologies or cannot be decomposed by the algorithm, the plant section must be divided into smaller sections and analyzed separately. For all SDG graph-based operations, the Python library NetworkX is used (cf. Hagberg et al., 2008). The implementation of the algorithm and the functions to perform graph-based analysis using Python are shown in Appendix Listing R-1, Appendix Listing R-2, and Appendix Listing R-3.

An example is used to illustrate the algorithm's procedure (cf. Figure 74). The plant section consists of a pump (1), piping (2), and control valve (3), which leads into a settling tank (4), and a nitrogen blanket (5), which also leads into the settling tank (4). The plant section downstream of the settling tank is not described further and is called sink (6). The SDG representation of the plant section is shown in Figure 75. From this illustration, it becomes apparent that the present plant section is a junction system (cf. Figure 70 c) because the node strands 0-4 and 6 end in node 5 (cf. Figure 75).

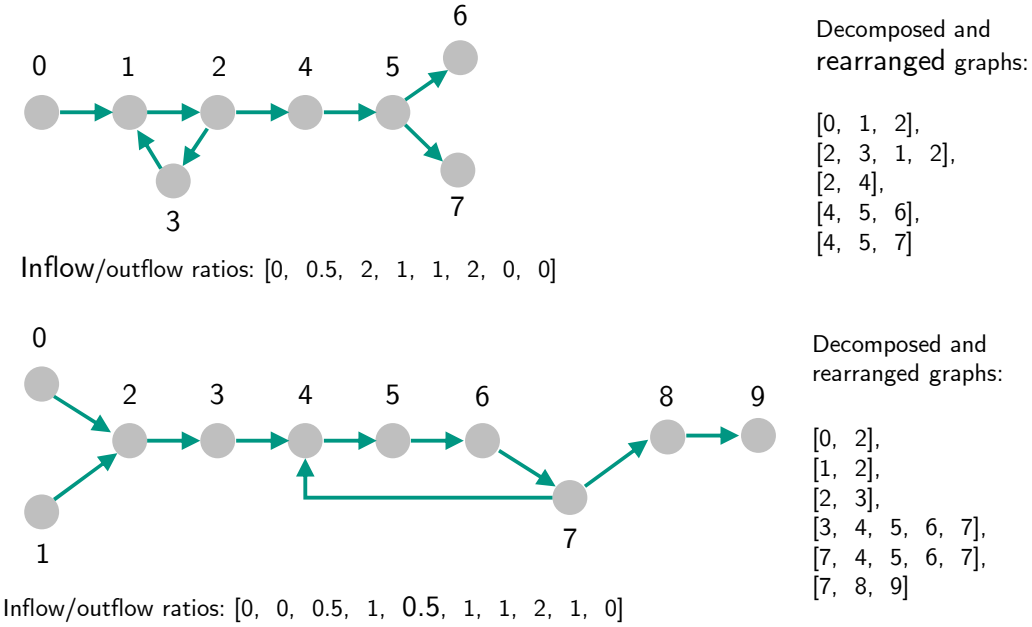


Figure 73: Flow chart of the plant section identification algorithm

Thus, events can propagate through both lines and end up in the settling tank. The algorithm's scope recognizes the 'junction topology' and rearranges the SDG representation for propagation. The SDG representation serves as an input for the algorithm. First, the SDG is examined for cycles, which can be identified with a build-in function of the NetworkX library (cf. Figure 70 d). If cycles are detected, another strategy must be used (cf. Appendix Figure R-6). Then, the outflow/inflow stream ratio is calculated for each node (cf. Appendix Figure R-7). The smallest ratio is identified regarding a 'junction layout' system, which is 1/2 (cf. Figure 75, node 5). For instance, if three streams enter and one stream leaves, the ratio is 1/3. Depending on the number of incoming streams, the outflow/inflow stream ratio of the junction node can be between $0 < r_k <= 0.5$. Besides, there is only one node with the smallest ratio within a junction layout. If both conditions are met (cf. Appendix Figure R-7), a junction layout is assumed. Afterward, the graph is rearranged into three lines evaluated separately (cf. Figure 75).

After identifying the type of topology and the decomposition into lines, deviations and effects are propagated through the lines. Within the junction, all propagated events are merged. The example in Figure 75 shows that the propagated events from line 1 and line 2 would end in the settling tank, where they may lead to other events.

5.14.3 Propagation of deviations using propagation stacks

Partial scenarios that qualify for propagation (cf. Figure 28) are stored in equipment-specific lists, called *propagation stacks* within this work (cf. Figure 76). Initially, the propagation stacks are filled in the equipment-based mode and processed in propagation-based mode (cf. Figure 69). Qualifying partial scenarios are propagated through the plant section in the propagation mode of the inference algorithm, resulting in further scenarios. Typically, the partial scenarios provide the causes for scenarios in adjacent equipment. After each equipment entity, it is determined whether partial scenarios lead to complete scenarios. Scenarios under consideration are propagated to the next equipment entity or are resolved. A purely equipment-based approach cannot identify the causes of specific scenarios since adjacent equipment is not considered. For example, Figure 76 shows a partial scenario within the inert gas blanketing entity, forming a complete scenario inside the settling tank. The scenario involves the 'no flow' deviation within an inert gas blanketing entity caused by a "NoInertGasSupply" with the underlying cause "UtilityFailure".

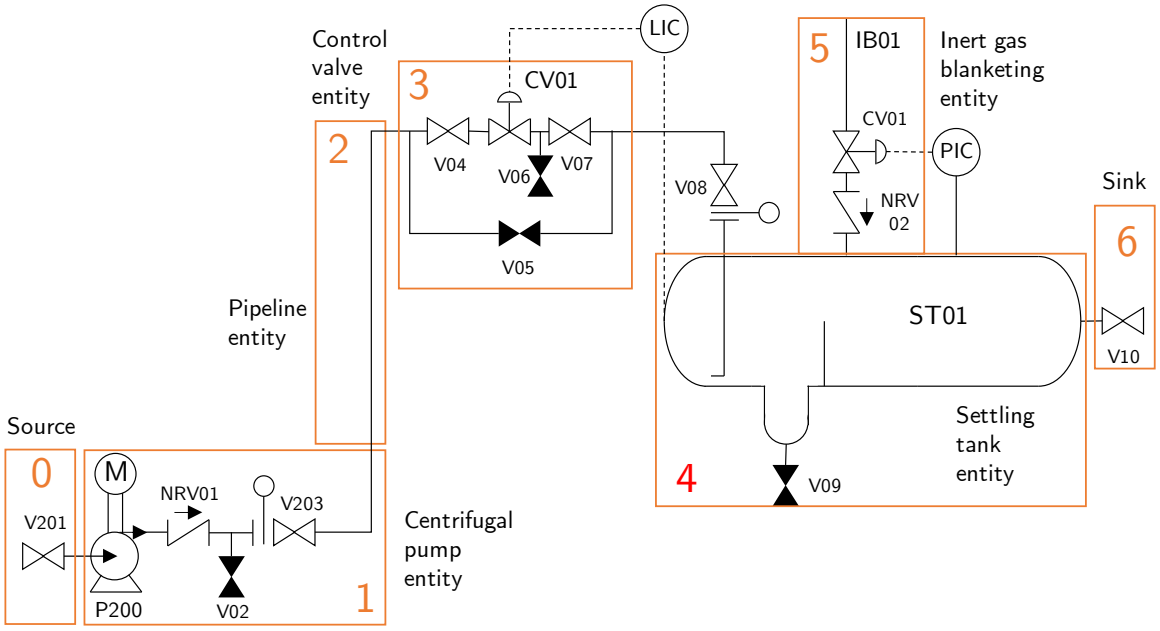


Figure 74: P&ID of the plant section

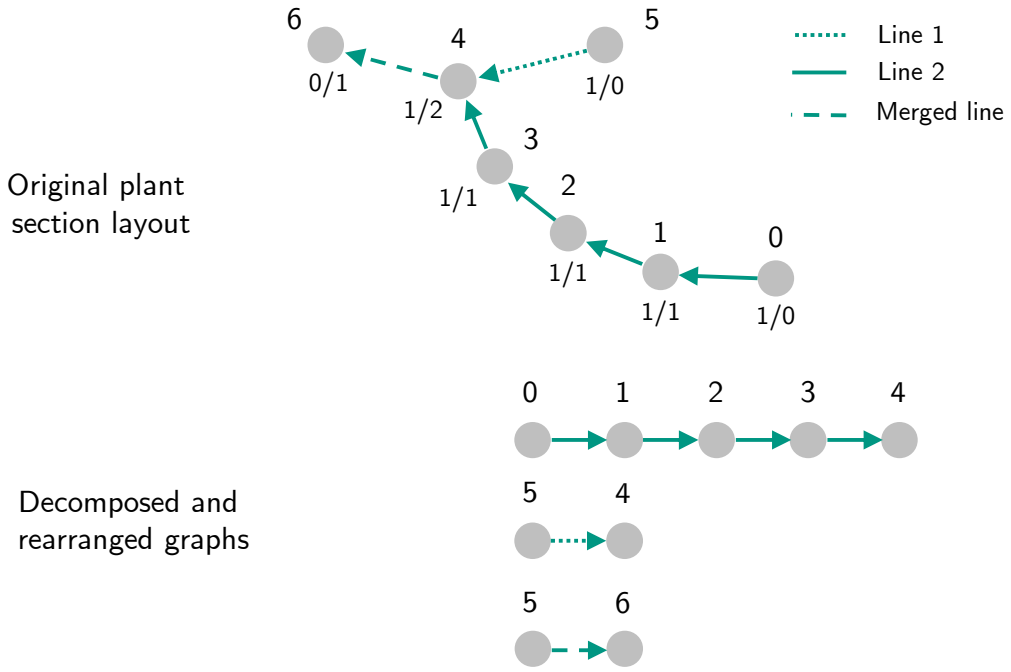


Figure 75: Graph representation of the original plant section and decomposed plant section

Since there are no immediate effects and consequences within the inert gas blanketing unit (*None* in Figure 76), this is a partial scenario that qualifies for propagation. This scenario is propagated into the settling tank since the settling tank is connected to the inert gas blanketing unit. Based on the described partial scenario, the effect of "InsufficientInertization", and the consequence "RiskOfExplosiveAtmosphere" are identified within the settling tank, making it a complete scenario. If deviations or partial scenarios are propagated through plant sections, propagation paths result, as shown in Figure 77. The settling tank from Figure 74 serves as an example, where water is separated from liquid olefin inside a settling tank overlaid with nitrogen. This results in different propagation paths, which can arise in different equipment entities and then propagate through plant sections. In this case, a contaminant from outside (**OtherThanComposition**) is propagated through all equipment entities into the settling tank. If the pump delivers a too low mass flow (**LowFlow**), the deviation **LowLevel** results in the settling tank. An incorrectly set or activated control valve (**HighFlow**) can lead to a **HighLevel** deviation with the subsequent deviation **HighPressure** in the settling tank. A **HighFlow** deviation in the inert gas blanketing unit can lead to a **HighPressure** deviation in the settling tank.

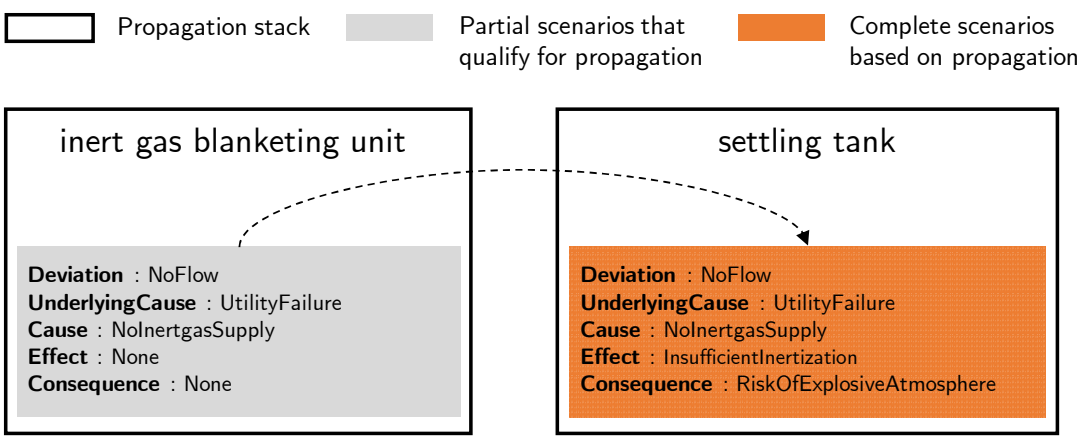


Figure 76: Example propagation of a partial scenario

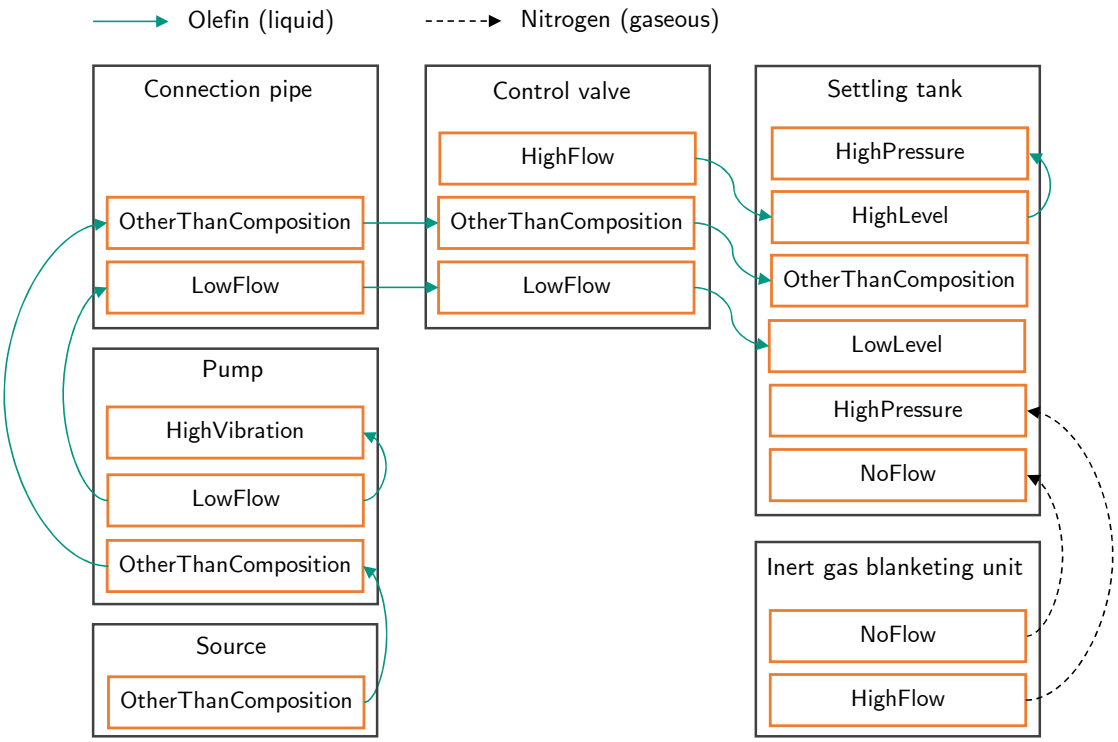


Figure 77: Example propagation paths within the settling tank system

5.15 The architecture of the developed knowledge-based system

The schematic architecture of the developed knowledge-based system and the interactions between user, knowledge engineer, and domain expert is shown in Figure 78. The main modules within the application logic are equipment-based analysis (cf. Section 5.12) and propagation-based analysis (cf. Section 5.14). Both modules exploit the ontology and the backward reasoning module employing the HermiT reasoner. Within the propagation mode, the topology of the plant section is additionally analyzed (cf. Section 5.14.1; 5.14.2). An automatically generated extended HAZOP worksheet is available as an HTML file. The ontology engineer conceptualizes, formalizes, maintains, and extends the ontology while the domain expert provides the technical input for representing the knowledge domain. In addition, the domain expert checks the correctness, completeness, and quality of the generated HAZOP worksheets and suggests changes to the knowledge engineer. The input object is the basis for every automated HAZOP, and the creation of the input object is described in Section 5.16.

5.16 Procedure for the application of the proposed method

For applying the proposed method and the developed prototype of the proposed knowledge-based, an input data object must be provided, containing all required information regarding plant, process, involved substances, and boundary conditions. Figure 78 shows the architecture of the knowledge-based system and the interactions with the user, knowledge engineer, and domain expert. The components of the input data object have been presented in Figure 49. Case studies illustrate further formal input data objects (cf. Chapter 6). The Python code of the main routine of the knowledge-based system is shown in Appendix C.

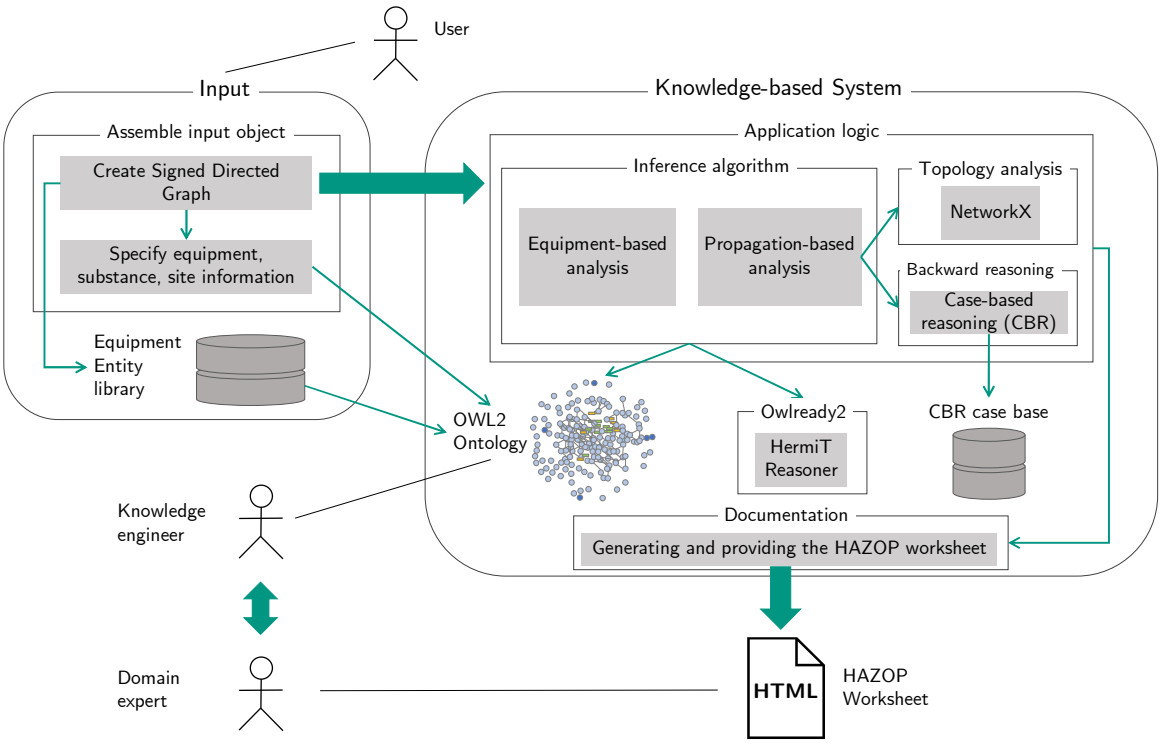


Figure 78: Architecture of the knowledge-based system and interaction with humans

The following steps must be performed to create the input object:

- First, the plant section under consideration (HAZOP nodes) must be isolated from the P&ID. Then the equipment of the plant section, including all involved modes of operation, intended function, fixtures, piping, instrumentation, subsystems, and the connections between the equipment, must be specified. The developed equipment entity model library can be used for this purpose (cf. Section 5.6; Figure 78).
- Based on a safety data sheet (SDS), a model of the substances involved is created (cf. Section 5.2.3). These substances must be assigned to each equipment entity. Different substances can be present in different equipment entities.
- Besides, environmental information can be provided, which involves environmental hazards. Additionally, site information, such as ambient temperature, and traffic (e.g., forklifts, trucks, rail tank wagon), can be provided (cf. Section 5.7).
- In addition, boundary conditions must be defined that further restrict the concepts and capture assumptions, e.g., 'personnel present in the near field' (cf. Section 5.5.8). Boundary conditions can be assigned equipment-specific or can be assigned globally (cf. Figure 49).
- Afterward, the knowledge-based system evaluates the overall model (cf. Section 5.7) consisting of process plant, substance representation, site information, and boundary conditions, identifies scenarios based on the ontology (cf. Section 5.8), identifies plant configurations within the propagation mode (cf. Section 5.13.2) and automatically generates HAZOP worksheets.
- If the HAZOP practitioner identifies scenarios in the automatically generated results that are incomplete or invalid, they can be revised by adjusting scenario components. Also, the concepts can be supplemented by further restrictions, or completely new concepts can be added to alter the inferred scenarios.

5.17 Summary of the developed knowledge-based system

A knowledge-based system was developed and investigated to digitize the HAZOP method (cf. Chapter 2, (3)). The underlying causation model of HAZOP typical scenarios was investigated and extended. The result of the ontology development process is a complete formal ontology-based model of the scenario components *underlying cause*, *cause*, *deviation*, *effect*, and *consequence*. In the conceptualization of the ontology, special attention was paid to considering semantic relationships between scenario components, likelihood, severity, risk, safeguards, structural and functional characteristics of equipment, substance properties, and site information. Due to the modeling of causal relationships between scenario components, it is possible to dynamically compose scenarios based on scenario components to avoid the *content completeness problem* and scaling problems (cf. Chapter 2, (3)). Formal DL-based ontologies were developed for representing interdisciplinary expert knowledge using the ontology language OWL 2. In the context of this research, several ontologies were created to become familiar with the technology and apply and investigate different styles of modeling and scopes. An evolutionary development approach was chosen during the development because the suitability of an ontology can only be investigated in the context of an evaluation algorithm. Ergo, ontology development and evaluation algorithm development are mutually dependent. The ontology was created programmatically using the Python package Owlready2. Besides, unit tests for specific ontology class definitions were conducted, simplifying debugging during development (cf. Chapter 2, (4)).

For the application of the developed method, an input data object is needed, which is processed by the knowledge-based system. An SDG-based approach was developed to model the topology of plant sections. Each node of the SDG contains information about the equipment entity, intended function, involved substances, site information, and all connections (edges) to adjacent equipment. A reusable equipment entity library was developed based on structural and functional characteristics of equipment to assemble plant sections (cf. Chapter 2, (5)). The information contained in safety data sheets is sufficient to characterize the involved substances adequately. In addition, site information, such as ambient temperature ranges, environmental hazards, and information about vehicles or cranes in the vicinity of the plant, are considered. All information is attached to the nodes of the signed directed graph (SDG) in the form of objects (cf. Chapter 2, (6)).

Based on identified scenario components, the developed knowledge-based system conducts a dynamic risk assessment, in which the risk class of individual scenarios is determined automatically. The risk assessment makes it possible to propose valid safeguards that interrupt the event sequence of the scenarios (cf. Chapter 2, (7)). An inference algorithm was developed to emulate the HAZOP method. The HermiT reasoner exploits the ontology to identify scenarios within forward reasoning steps. Case-based reasoning is used in a backward reasoning step for more complex scenarios to identify subsequent process deviations and subsequent scenarios based on partial scenarios. The order and systematics in which different deviations, substances, modes of operations, intended functions are processed within the inference algorithm are crucial for the reasoning performance, quality of achievable HAZOP worksheets, and the runtime during automatic evaluation (cf. Chapter 2, (8)). A distinction is made between the equipment and propagation-based strategies within the inference algorithm. Equipment entities are considered individually during the equipment-based analysis without considering adjacent equipment entities. Process deviations and effects are propagated through the plant section under consideration of the plant topology to identify scenarios across equipment entities in the propagation mode. The propagation stack method was developed to identify propagating partial scenarios. (cf. Chapter 2, (9)).

6. Validation of the developed procedure

In the following chapter, the proposed method and developed prototype of the knowledge-based system consisting of the latest ontology (August 2021), inference algorithm, and the equipment entity library is applied in different case studies. The goal is to automatically generate HAZOP worksheets and analyze the quality and quantity of the results. In addition to discussing the results generated, direct comparisons are made with conventionally created HAZOP worksheets typical for the industry. The case studies are selected to demonstrate different aspects of using the developed knowledge-based system. In the first case study, the focus is on considering different intended functions and showing the procedure and the method in detail. The second case study involves multiple substances and a junction topology.

6.1 Case study I: Hexane storage tank

The first case study involves a feed line and a storage tank that stores liquid hexane. The hexane tank example was adapted from (CCPS, 2001). Because sufficient reference data is available, it is suitable for demonstrating the developed method. The case study was already presented in excerpts in (cf. Single et al., 2020d).

The tank is regularly filled with hexane through a tank truck via the feed line, including an unloading pump. According to the original process description, two operators initiate the hexane transfer from the tank truck into the storage tank and monitor the process in the control room. Hexane is usually used as a solvent or reaction medium. The tank is filled to a maximum of two-thirds during normal operation. As the supply line is under pressure, it is assumed that the tank's operating pressure is higher than the ambient pressure. No further specifications are made regarding the pressure in the tank. A schematic P&ID of the system under consideration is shown in Figure 79. The downstream equipment and process of the storage tank are not considered within this case study. No further protective measures, design, or procedural safeguards other than the dike are considered but should be identified by the knowledge-based system.

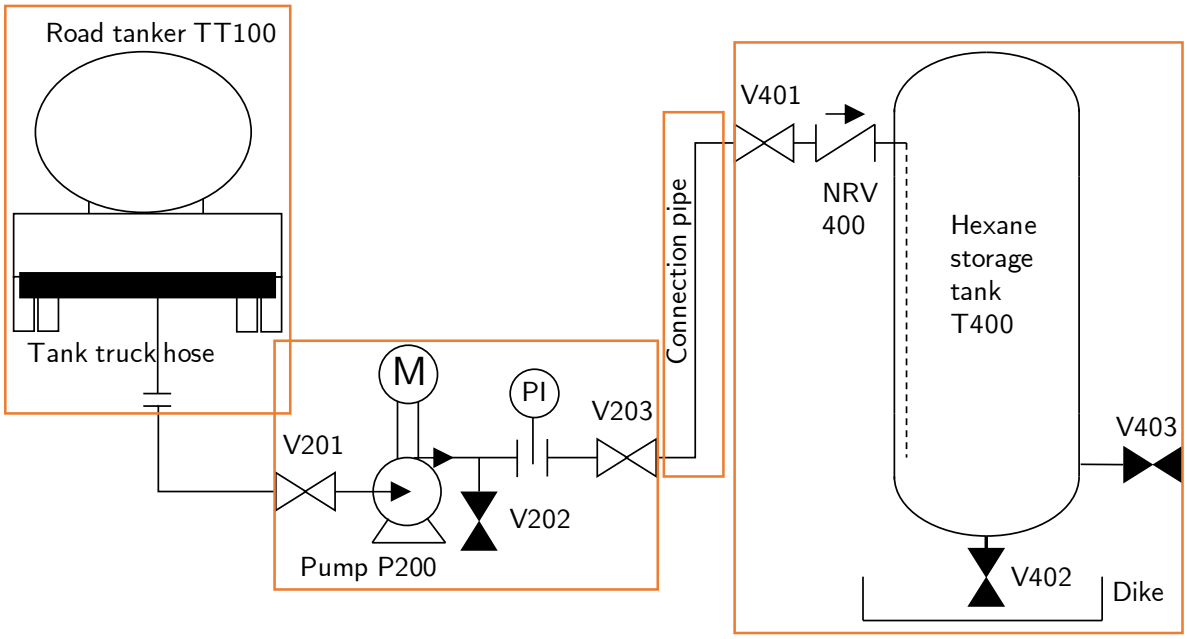


Figure 79: P&ID of the Hexane tank

6.1.1 Model representation of the hexane tank

A valid input data object is required to apply the developed method. In the original literature, the road tanker is specified to have a capacity of 50000 lb, and the storage tank a capacity of 80000 lb. Using the density (cf. Table 18), the volume of the road tanker is 34.4 m^3 while it is 55 m^3 of the storage tank. The transfer of hexane is conducted through an unloading pump controlled manually by an operator. Since the original literature does not provide any information on the maximum volume flow, a volume flow of 90 m^3/h is specified. The considered equipment, including all considered plant items, are shown in Figure 80.
Furthermore, the involved substance and its properties are modeled using a Safety Data Sheet (SDS) of hexane. Relevant sections of the SDS are shown in Table 18, while the entire SDS was downloaded from DHC (2017). The liquid hexane is delivered from the tank truck, through the feed line, and into the hexane tank. Based on the information in the SDS and the state of aggregation of the substance involved, the required substance entity object can be composed. An object diagram in UML notation of the substance entity object is shown in Figure 80. Since multiple substances can be located within the equipment, the substance is associated with each equipment entity (cf. Figure 80). Appendix Listing W-1 shows the Python function for creating the input data object of the hexane tank. A location in Germany is considered, where a maximum temperature of 35 °C and a minimum temperature of -8 °C is present. Further boundary conditions are that the hexane tank is standing outside, that impurities can be introduced into the tank, that personnel is present in the near field of the tank, and that an external fire can occur. This results in the following boundary conditions, which are considered:

- **LocatedOutside → SufficientOxygenAvailable**,
- **IntroductionOfWater**,
- **PersonnelPresentInTheNearField**,
- **ExternalFirePossible**,
- **IntroductionOfImpurities**.

Table 18: Excerpt from the Safety Data Sheet: Hexane

Hexane (n-Hexane)			
Lower explosion limit	1.1 vol%	**Upper explosion limit**	8.3 vol%
Odor	characteristic	**Vapor pressure**	20-30 kPa @ 20 °C
pH-value	Not determined	**Density**	0.66-0.68 g/cm^3 @ 15 °C
Melting/Freezing point	-95 °C	**Viscosity**	0.47-0.55 mm^2/s @ 20 °C
Boiling point/Boiling range	65 °C	**Auto/Self-Ignition temperature**	>220 °C
Flash point (closed cup)	-20 °C	**Decomposition temperature**	Not determined

Hazard class	**Category**
Flammable liquid	Cat. 2
Skin corrosion/irritation	Cat. 2
Reproductive toxicity	Cat. 2
Specific target organ toxicity - single exposure	Cat. 3
Specific target organ toxicity - repeated exposure	Cat. 2
Aspiration hazard	Cat. 1
Hazardous to the aquatic environment - chronic	Cat. 2
Stability and reactivity information	
Incompatible with oxidizers	Can form explosive mixtures with air

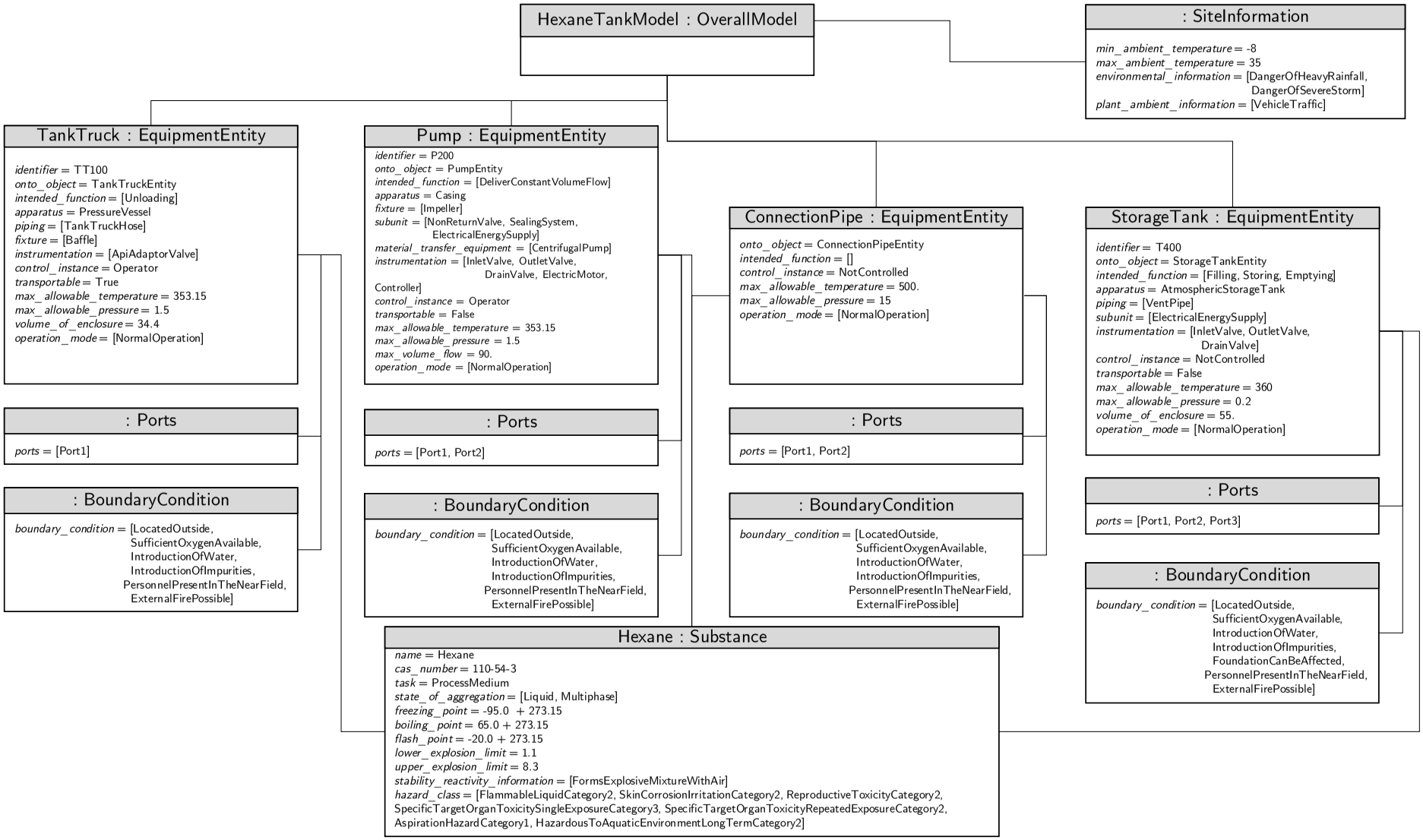


Figure 80: Object diagram of the hexane storage tank model

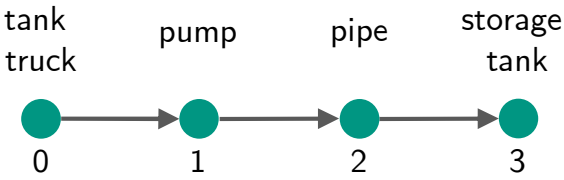


Figure 81: SDG representation of the tank truck, pump and hexane tank

The input object represents the system model, which is represented in an object-oriented manner based on the class definitions of the ontology. The connections between the equipment are presented in the form of an SDG, which is relatively simple because it is a single stream system (cf. Figure 81). The hose that connects the tank truck with the pump is considered part of the tank truck and is not modeled separately. The different intended functions and operating modes of the individual equipment entities are shown in Table 19.

6.1.2 Automatically generated HAZOP results regarding the hexane tank

The application of the described and developed method provides automatically generated extended HAZOP worksheets. These worksheets are generated by the computer code implemented within Python and are returned as Hypertext Markup Language (HTML) files that can be further processed (cf. Single et al., 2020d). These are extended worksheets because the extended description of the scenario by the concepts **UnderlyingCause** and **Effect** adds two more columns. Also, there are the columns equipment entity, equipment id, mode of operation, and substance. In classical worksheets, these columns do not exist. Substance and equipment-specific details are usually described in the node description. The additional columns in the automatically generated worksheets can be explained because the computer system sequentially runs through all equipment entities and considers all involved substances and operation modes. There are no references to other scenarios within the generated HAZOP worksheets since every scenario is fully described.

The automatically generated HAZOP worksheets are shown in Table 20 and Table 21. For the sake of clarity, the cells of identical entries are merged, yet different scenarios can be distinguished by their numbers. If no specific cell entry is identified within the worksheets, the cell is marked with a dash ('-'). This is not an incomplete scenario per se because, for example, the underlying cause concept is optional. In general, all listed scenarios differ due to at least one characteristic, even if scenarios appear similar.

The results were generated by combining an equipment-based with a propagation-based analysis. The integral behavior across the HAZOP node is analyzed within the propagation algorithm, which provides further results and explanations (cf. Section 5.14). In the propagation case, the HAZOP table contains explanations in the **Cause** column, which describe the origin equipment, the initiating event, and the resulting deviation. Additionally, there is a reference to the considered state of aggregation, which is relevant if several states of aggregation exist. The propagation is only related to one of them (cf. Table 20, Pos. 12-17). All scenarios containing an explanation in the form "P200: ClosedInletValve → PumpRunningDry → HighTemperature (Liquid)" in the **Cause** column were identified using a combination of forward and backward reasoning (cf. Section 5.13).

Table 19: Function description matrix of the hexane tank

Type	Tank truck	Pump	Connection pipe	Storage tank
IntendedFunction	ModeIndependent	ModeIndependent	ModeIndependent	ModeIndependent
	Unloading	DeliverVolumeFlow	-	Filling
	-	-	-	Storing
	-	-	-	Emtying
Operation mode	NormalOperation	NormalOperation	NormalOperation	NormalOperation

Overall, 55 scenarios involving 11 different deviations were identified automatically. Concerning the tank truck, the intended function **Unloading,** and the centrifugal pump, the intended function **DeliverConstantVolumeFlow** are given. Since the connection pipe only has structural properties, there is no intended function, and all scenarios are identified **ModeIndependent**.

Concerning the tank truck, regarding the intended function **Unloading**, only scenarios related to the leakage of hexane occurred, which leads to a "PoolFormation", and "LossOfPrimaryContainment", which in turn lead to the subsequent consequences "FireHazard" or "RiskOfExplosiveAtmosphere". The subsequent consequences are potential consequences because both consequences require an ignition source. Per se, no explosion or fire can be assumed. The detailed evaluation of the consequences is performed in the context of the consequence analysis, such as dispersion calculation. Different underlying causes such as "LossOfLeakTightness"; "BrokenHose", and "HoseIncorrectlyConnected" are identified as the cause of the external leakage (cf. Table 20, Pos. 1-3).

Overall, 15 pump-related scenarios regarding the feed line are identified, both safety-relevant and bear potential operability issues. Within the underlying causes column, human errors like "OperationalError" or "MaintenanceError" are listed, which are the causes of incorrect operating conditions or inadvertently closed or opened valves. Faulty operating states can result in scenarios that are relevant to safety. For example, the effects "PumpRunningDry" or "Cavitation" can lead to the deviation **HighVibration**, which can, in turn, lead to a "FatigueFracture" over time, which potentially leads to LOPC of hazardous hexane (cf. Table 20, Pos. 15-17).

Scenario 11 in Table 20 describes the pump-related scenario involving a "PoolFormation" expressed by the deviation **ElsewhereFlow** and the cause "PumpSealFailure". The pump seal failure may have the underlying cause "WearDown". Based on the hazard classes of hexane, the risk category **B** results for the pump seal failure scenario. Scenarios 22 and 23 in Table 20 are identified in propagation mode (cf. Section 5.14), indicated by 'True' in the propagated column. A too high flow velocity (deviation: **HighFlow**) in the pipeline caused by a "PumpIncorrectlySet" or "WrongImpeller" may lead to the effect "EmergenceOfIgnitionSource", which is a considerable hazard inside the tank due to the flammability of hexane, which results in the risk class **B** (cf. Table 20, Pos. 22-23). To prevent "EmergenceOfIgnitionSource", the safeguards "ConsiderMaterialSelection", "LimitationOfTheFlowVelocity", and "ProvideGroundingOfPlant" are identified (cf. Table 20, Pos. 22-23).

Another safety-relevant scenario involving the centrifugal pump is related to the cause "DeadHeadingOfPump" and the effects "FluidCirculatesInsidePump", and "Overheating", which can pose a danger for a boiling liquid expanding vapor explosion (BLEVE) (cf. Table 20, Pos.7). For this scenario, the risk category **B** was identified, which results from the automatically identified likelihood **Occasional**, and the severity **Severe** (cf. Appendix Listing O-3).

These examples show the importance of risk-ranking scenarios because otherwise, the significance of individual scenarios is incorrectly assessed. The identified risk class is the unmitigated risk, i.e., no protective measures, physical or organizational safeguards are considered, which is typical in the chemical process industries.

Scenarios 8-10 in Table 20 describe operability problems and have no direct safety implications but may disrupt or interrupt operation. The effect of "IncreasedWear" leads to the potential consequences of "PumpBreakdown" and "ProductionDowntime" (cf. Table 20, Pos. 8-9). Also, the operational problem "Cavitation" is detected, which can be traced back to the cause "InsufficientNpsh" or the deviation **LowPressure** resulting in a "PumpBreakdown" and a "ProductionDowntime" (cf. Table 20, Pos. 10). The risk of scenarios that concern operability issues are classified here as risk class **D** due to the low severity of these scenarios. For example, a "PumpBreakdown" is rated with the severity category **Minor** (cf. Appendix Listing O-3).

Table 20: Automatically generated extended HAZOP worksheets of the hexane tank I

Pos.	Equipment entity	Id.	Mode / Intention	Substance	Deviation	Underlying cause	Cause	Effect	Consequence	Safeguard	Propagated	Risk
1	TankTruckEntity	TT100	Unloading	Hexane	ElsewhereFlow	HoseIncorrectly-Connected	ExternalLeakage	PoolFormation	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	DetermineSafetyRelatedOperatingInstructions, ElaborationOfMaintenanceConcept, RegularPlantInspection	False	B
2						LossOfLeakTightness				ElaborationOfMaintenanceConcept, RegularPlantInspection	False	B
3						BrokenHose					False	B
4	PumpEntity	P200	DeliverConstant-VolumeFlow		NoFlow	MaintenanceError	MissingImpeller	PumpDeliversNoLiquid	ProductionDowntime	ImplementNoFlowWarning, StandardOperationProcedure	False	D
5						ValveFailure	ClosedInletValve	PumpRunningDry	PumpBreakdown, ProductionDowntime	DryRunProtection, StandardOperationProcedure, ValveLockedOpen	False	D
6						OperationalError					False	D
7							DeadHeadingOfPump	FluidCirculatesInsidePump, Overheating	DangerOfBleve	ImplementNoFlowAlarm, OverFlowValveAndKickBackLine, OverflowValve, ThermalProtectionTrip, ValveLockedOpen	False	B
8					LowFlow	-	ImpellerFault	IncreasedWear	PumpBreakdown, ProductionDowntime	LowFlowProtectionTrip, StandardOperationProcedure	False	D
9						-	OperationBelowMinimumFlowRate				False	D
10					LowPressure	-	InsufficientNPSH	Cavitation		IncreaseSuctionPressure, InstallPumpInducer	False	D
11					ElsewhereFlow	WearDown	PumpSealFailure	PoolFormation	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	ElaborationOfMaintenanceConcept, RegularPlantInspection	False	B
12					HighTemperature	ValveFailure	P200: ClosedInletValve -> PumpRunningDry -> HighTemperature (Liquid)	Overheating	PumpBreakdown, ProductionDowntime	OverFlowValveAndKickBackLine, StandardOperationProcedure, ThermalProtectionTrip	False	D
13						OperationalError					False	D
14						OperationalError	P200: DeadHeadingOfPump -> FluidCirculatesInsidePump -> HighTemperature (Liquid)				False	D
15					HighVibration	ValveFailure	P200: ClosedInletValve -> PumpRunningDry -> HighVibration (Liquid)	FatigueFracture	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	ImplementVibrationDampener, RegularPlantInspection	False	B
16						OperationalError					False	B
17						-	P200: InsufficientNPSH -> Cavitation -> HighVibration (Liquid)				False	B
18	ConnectionPipeEntity	PIP1	ModeIndependent		HighPressure	RapidlyClosingValve	WaterHammer	Fracture, PressureExceedingDesignPressure		CheckPressureClassPiping, IncreaseClosingTimeOfValve, PulsationDampener, RegularPlantInspection, SurgeReliefValve, SwingCheckValve	False	B
19						BlockedPiping-AndHeatInput	ThermalExpansion	Fracture, PressureExceedingDesignPressure		CheckPressureClassPiping, InstallThermalExpansionReliefValve, RegularPlantInspection	False	C, B
20					ElsewhereFlow	LossOfLeakTightness	ExternalLeakage	PoolFormation		ElaborationOfMaintenanceConcept, InstallCheckValve, RegularPlantInspection	False	B
21					HighVibration	RapidlyClosingValve	PIP1: WaterHammer -> HighVibration (Liquid)	FatigueFracture		RegularPlantInspection	False	B
22					HighFlow	OperationalError	P200: WrongImpeller -> HighFlow (Liquid)	GenerationOfElectrostaticCharge	EmergenceOfIgnitionSource	ConsiderMaterialSelection, LimitationOfTheFlowVelocity, ProvideGroundingOfPlant	True	B
23						OperationalError	P200: PumpIncorrectlySet -> HighFlow (Liquid)				True	B
24	StorageTankEntity	T400	Filling		HighLevel	OperationalError	ExcessiveInflow	Overfilling	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	DefineMaximumFillLevel, FlameArrestor, InstallHighLevelAlarm, OverfillProtection, RegularPlantInspection, StandardOperationProcedure	False	C, B
25						FailureControlLoop	IncorrectIndicationOfFillingLevel			DefineMaximumFillLevel, FlameArrestor, OverfillProtection, RegularPlantInspection	False	B
26					LowLevel	DepositionOfImpurities	BlockedInflowLine	InsufficientFilling	ProductionDowntime	ImplementLowLevelWarning	False	D
27						-	LossOfInflow				False	D
28					HighPressure	AbnormallyHotIntake	T400: AbnormalHeatInput -> AbnormalEvaporation -> HighPressure (Multiphase)	Fracture, PressureExceedingDesignPressure	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	ConsiderMinimumTankDistance, FlameArrestor, PressureVacuumReliefValve, RegularPlantInspection	False	C
29						SolarRadiation					False	C, B
30						ExternalFire			LossOfPrimaryContainment, FireHazard	ConsiderMinimumTankDistance, EmergencyPressureReliefValve, PressureVacuumReliefValve, RegularPlantInspection	False	B
31						OperationalError	T400: ExcessiveInflow -> VolumetricDisplacement -> HighPressure (Multiphase)	Fracture, PressureExceedingDesignPressure	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	ConsiderMinimumTankDistance, FlameArrestor, PressureVacuumReliefValve, RegularPlantInspection	False	B

Safeguards such as “DryRunProtection”, “IncreaseSuctionPressure”, “ValveLockedOpen”, and “InstallPumpInducer” are suggested to ensure safe pump operation (cf. Table 20, Pos. 5-6 & 10).

Shutting off the inflow line in conjunction with an external heat input (e.g., solar radiation) can lead to an impermissible pressure build-up (cf. Table 20, Pos. 19). In addition, the sudden closing of a valve may cause a water hammer in the inflow line (cf. Table 20, Pos. 18).

There are three different intended functions concerning the storage tank: **Filling**, **Storing**, and **Emptying**. The scenarios during filling are mainly caused by faulty pump operation in the feed line or by operator errors. A very typical storage tank scenario is shown in Table 20, Pos. 24-25. The scenario in which an “ExcessiveInflow” or “IncorrectIndicationOfFillingLevel” leads to an “Overfilling”, which leads to the potential consequences “FireHazard” and “RiskOfExplosiveAtmosphere” due to hazardous attributes of hexane. “OperationalError” and “FailureOfControlLoop” were identified as underlying causes, which leads to the risk category **B** and **C/B** for the scenarios (cf. Table 20, Pos. 24-25).

The effect “VolumetricDisplacement” of the gas phase in the tank can be traced back to a high inflow of the pump caused by “PumpIncorrectlySet” or “WrongImpeller”. The result of it is pressure increase (deviation: **HighPressure**) in the storage tank, which can lead to a “PressureExceedingDesignPressure” and a subsequent “Fracture” in the storage tank with the consequence “LossOfPrimaryContainment” (cf. Table 21, Pos. 54-55). These scenarios were automatically ranked with the risk category **B** since every tank filling causes volumetric displacement.

The ‘other than composition’ deviation contains scenarios that can occur directly during filling or storage operation. Contamination of the hexane by water occurs during the filling phase and can have the causes “InadvertentContamination”, or “ContaminationOfUnloadingLines”, which can lead to the effect “AccumulationOfImpurities” (cf. Table 21, pos. 48-50).

For eleven scenarios, two different risk classes were identified (cf. Table 20, Pos. 19, 24, 29; Table 21, 32-33, 36, 45-46, 49). Since all class definitions are based on formal logic and corresponding set theory, intersections between classes can occur. The likelihood and severity categories from which the risk class is composed are based on different components of the scenario, the substance properties, and site information. Due to the combinatorial possibilities, overlapping the identified risk category resulting in two risk categories is possible (cf. Section 7.2.4).

Since the downstream process is outside the system boundary and not part of the HAZOP node, a single scenario was identified for the operation mode **Emptying** (cf. Table 21, Pos. 32). The transfer of liquid out of the storage tank without sufficient air volume to follow leads to a vacuum (deviation: **LowPressure**) forming in the tank, and a “LossOfMechanicalIntegrity” can occur. The mitigative safeguard “ConsiderMinimumTankDistance” and “PressureVacuumReliefValve” to mitigate the impact on adjacent tanks or plants, “GasBalanceLine” to prevent the scenario, and the general measures “FlameArrestor” and “RegularPlantInspection” were identified.

Further scenarios are not necessarily dependent on the intended function. For example, the scenario “ExternalLeakage” caused by corrosion resulting in a “LossOfPrimaryContainment” with potential subsequent hazards is independent (cf. Table 21, Pos. 41). In addition, events such as “MechanicalFailureOfSupport” with a subsequent “LossOfMechanicalIntegrity” or a “PhysicalImpact” due to a “VehicleCollision” with a subsequent “Fracture” have been identified (cf. Table 21, Pos. 42-43). The failure of the foundation (“MechanicalFailureOfSupport”), e.g., by soil mechanical processes or the penetration of substances, is a rare event with a severe impact expressed by the risk category **C**. A physical impact, e.g., by collision with a vehicle, leading to a fracture, is more likely to occur than the mechanical failure of the support. Thus, the identified risk category of the vehicle collision scenario is **C/B** (cf. Table 21, Pos. 42-43).

Table 21: Automatically generated extended HAZOP worksheets of the hexane tank II

Pos.	Equipment entity	Id.	Mode / Intention	Substance	Deviation	Underlying cause	Cause	Effect	Consequence	Safeguard	Propa-gated	Risk
32	StorageTank-Entity	T400	Emptying	Hexane	LowPressure	-	LiquidTransferWithoutCompensation	LossOfMechanicalIntegrity	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	ConsiderMinimumTankDistance, FlameArrestor, GasBalanceLine, PressureVacuumReliefValve, RegularPlantInspection	False	A, B
33			Storing		HighPressure	SolarRadiation	InsufficientThermalOutbreathing	Fracture, PressureExceedingDesignPressure		ConsiderMinimumTankDistance, FlameArrestor, PressureVacuumReliefValve, RegularPlantInspection	False	A, B
34					LowPressure	-	ExcessiveFluidWithdrawal	LossOfMechanicalIntegrity			False	B
35						AmbientTemperatureChange	InsufficientThermalInbreathing				False	B
36					LowTemperature, OtherThanComposition	-	ContaminationByWaterAndTemperatureFallsBelowFreezingPoint	DrainlineFracture		FlameArrestor, InstallHeatTracingSystem, RegularPlantInspection	False	C, B
37					OtherThanComposition	Condensation-AirHumidity	ContaminationByWater	BacteriaGrowth	PoorProductQuality	PeriodicalSampleTaking	False	D
38					HighCorrosion		T400: ContaminationByWater -> BacteriaGrowth -> HighCorrosion (Multiphase)	PoolFormation	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	AddCorrosionInhibitor, ConsiderMaterialSelection, FlameArrestor, ProvideAntiCorrosionCoating, ProvideLeakageMonitoring, RegularPlantInspection	False	B
39			ModeIndependent			Contamination-InTankTruck	T400: InadvertentContamination -> BacteriaGrowth -> HighCorrosion (Multiphase)			AddCorrosionInhibitor, ConsiderMaterialSelection, FlameArrestor, PeriodicalSampleTaking, ProvideAntiCorrosionCoating, ProvideLeakageMonitoring, RegularPlantInspection	True	B
40						Introduction-OfRainwater	T400: ContaminationInUnloadingLines -> BacteriaGrowth -> HighCorrosion (Multiphase)			AddCorrosionInhibitor, ConsiderMaterialSelection, FlameArrestor, ProvideAntiCorrosionCoating, ProvideLeakageMonitoring, RegularPlantInspection, UseSealingCaps	True	B
41						LossOfLeak-Tightness	ExternalLeakage			AddCorrosionInhibitor, ConsiderMaterialSelection, ElaborationOfMaintenanceConcept, FlameArrestor, ProvideAntiCorrosionCoating, ProvideLeakageMonitoring, RegularPlantInspection	False	B
42					ElsewhereFlow	-	MechanicalFailureOfSupport	LossOfMechanicalIntegrity		ConsiderMinimumTankDistance, FlameArrestor, PressureVacuumReliefValve, RegularPlantInspection	False	C
43						VehicleCollision	PhysicalImpact	Fracture		ConsiderMinimumTankDistance, FlameArrestor, InstallPhysicalBarrierAroundTheEquipment, RegularPlantInspection	False	C, B
44						OperationalError	DrainValveInadvertentlyOpened	PoolFormation		FlameArrestor, ProvideLeakageMonitoring, RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B
45						OperationalError	LeakingDrainValve				False	C, B
46						MaintenanceError					False	C, B
47						LossOfLeakTightness				ElaborationOfMaintenanceConcept, FlameArrestor, ProvideLeakageMonitoring, RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	C, B
48					OtherThanComposition	IntroductionOfRainwater	TT100: ContaminationInUnloadingLines -> OtherThanComposition (Liquid)	AccumulationOfImpurities	PoorProductQuality	UseSealingCaps	True	D
49						OperationalError	TT100: WrongTankLinedUp -> OtherThanComposition (Liquid)	Fracture, PotentialViolentReactionWithOxidizers, PressureExceedingDesignPressure	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	ConsiderMinimumTankDistance, DetermineSafetyRelatedOperatingInstructions, FlameArrestor, ImplementQuickConnectSystem, RegularPlantInspection, StandardOperationProcedure	True	C, B
50						Contamination-InTankTruck	TT100: InadvertentContamination -> OtherThanComposition (Liquid)	AccumulationOfImpurities	PoorProductQuality	PeriodicalSampleTaking	True	D
51					NoFlow	MaintenanceError	P200: MissingImpeller -> NoFlow (Liquid)	InsufficientFilling	ProductionDowntime	StandardOperationProcedure	True	D
52						ValveFailure	P200: ClosedInletValve -> NoFlow (Liquid)			StandardOperationProcedure, ValveLockedOpen	True	D
53					HighFlow	OperationalError	P200: WrongImpeller -> HighFlow (Liquid)	Overfilling	LossOfPrimaryContainment, FireHazard, RiskOfExplosiveAtmosphere	DefineMaximumFillLevel, FlameArrestor, OverfillProtection, RegularPlantInspection	True	B
54					HighPressure	OperationalError	T400: WrongImpeller -> VolumetricDisplacement -> HighPressure (Multiphase)	Fracture, PressureExceedingDesignPressure		ConsiderMinimumTankDistance, FlameArrestor, PressureVacuumReliefValve, RegularPlantInspection	True	B
55							T400: PumpIncorrectlySet -> VolumetricDisplacement -> HighPressure (Multiphase)				True	B

The scenario involving "VehicleCollision" can be prevented by placing a physical barrier around the tank ("InstallPhysicalBarrierAroundTheEquipment"). These rather general measures are not documented in conventional studies by a HAZOP team if there is no traffic nearby.

Concerning the hexane storage tank, the safeguards aim at preventing an uncontrolled release of the flammable hexane, including subsequent incidents. Some identified safeguards are general, including the safeguard "RegularPlantInspection" to detect leakages or a "PoolFormation" at an early stage, and a "FlameArrestor" for preventing the propagation of flames. This may also refer to regulations, such as the German Technical Regulations for Hazardous Substances (TRGS 509), requiring a minimum distance between storage tanks to be maintained when the flashpoint falls below certain limits. Accordingly, based on the flashpoint of the involved substance, a minimum distance between storage tanks ("ConsiderMinimumTankDistance") is identified.

6.1.3 Reference HAZOP worksheet of the hexane tank

The original HAZOP results are used as a reference to compare the quality of the automatically generated results. Within the original HAZOP worksheets, a distinction was made between the feed line (cf. Table 22) and the hexane tank (cf. Table 23). The feed line includes the pipeline from the tank truck through the unloading pump to the storage tank. Process deviations were used as causes and consequences (cf. Table 23, Pos. 4.1 and 4.5). If no causes are identified, this is indicated by the entry 'no credible causes identified', while some cells are left blank without further explanation. If no dangerous or operational consequences have been identified, this is expressed by the 'No consequences of interest' entry. Also, the original HAZOP table uses cross-references (cf. Section 5.13). The automatically created worksheets of the proposed method are compared to the original HAZOP worksheets. In Table 22 and Table 23, scenarios are color-coded to indicate recognized, indirectly recognized, and scenarios for which other conclusions were drawn.

6.1.4 Differences and similarities in content and quality

Table 24 and Table 25 show that all safety-relevant scenarios presented in the original worksheets were identified directly or indirectly. Indirect means that the same conclusions were drawn in other scenarios or equivalent scenario components and measures were identified. The direct comparison of HAZOP results requires the interpretation of scenarios. Different deviations may lead to similar scenarios, which can be interpreted differently with the same consequences. Thus, The risk assessment was not performed within the original HAZOP worksheets. In the original worksheets, no explicit distinction was made between different operating states since these are automatically taken into account by the HAZOP conductors based on their experience. On the one hand, an explicit distinction between different operating states simplifies the plausibility check of the automatically identified scenarios. On the other hand, this makes it possible to restrict class definitions by the operation mode to analyze batch processes.

In the original HAZOP, the consequence of the 'high level' deviation is 'high pressure' (cf. Table 23, Pos. 4.1). Due to the clear semantic distinction between the concept of deviation and consequence, such a scenario description is not possible with the proposed method. Accordingly, the automatically generated worksheets the direct consequence of a 'high level' scenario is "Overfilling" (cf. Table 20, Pos. 24-25). In contrast to the original worksheets, scenarios were identified for the high flow deviation (cf. Table 22, Pos. 3.1). On the one hand, for a 'high flow' to a "VolumetricDisplacement" of the gas phase, which can lead without sufficient venting to an overpressure in the storage tank, to the "PressureExceedingDesignPressure", "Fracture", and a "LossOfPrimaryContainment" (cf. Table 21, Pos. 54-55). On the other hand, an increase in flow velocity in the feed line can lead to a "GenerationOfElectrostaticCharge", resulting in an "EmergenceOfIgnitionSource", which is a severe scenario for a flammable substance (cf. Table 20, Pos.

22-23). With the help of the presented knowledge-based system, scenario components are propagated through the equipment entities of a plant section.

Table 22: Reference HAZOP worksheet of the inflow line based on (CCPS, 2001)

Color legend	Recognized	Indirectly recognized	Other conclusion

Pos.	Deviation	Cause	Consequence	Safeguard
3.1	High flow	-	No consequences of interest	-
3.2	Low/no flow	Blocked flow (e.g., plugged line)	Potential overheating and failure of pump seal (see 3.9)	-
		Downstream manual block valve inadvertently closed or gate falls	Low level – hexane storage	-
		Low pressure (see 3.7)		
3.3	Reverse flow	Drain valve inadvertently left open while unloading pump is off	Possible loss of containment (see 3.9)	Check valve
		Low pressure (see 3.7)		
3.4	High temperature	No credible causes identified	-	-
3.5	Low temperature	-	No consequences of interest	-
3.6	High pressure	-	No consequences of interest	-
3.7	Low pressure	Unloading pump fails off	Low flow or no flow (see 3.2)	Local pressure gauge
			Reverse flow (see 3.3)	
3.8	High concentration of contaminants	Contamination (organic, moisture, or debris) in flexible unloading lines	High concentration of contaminants – hexane storage tank T-301 (see 4.7)	Hexane unloading procedure
		Contamination in the tank truck		Caps for flexible unloading line
		Receiving or spotting the wrong tank truck		Material testing procedure prior to unloading
3.9	Loss of containment	Corrosion/erosion	Release of hexane; fire hazard affecting a large area	Operation/maintenance response as required, including isolation if needed
		External fire		
		External impact		
		Gasket, packing or seal failure		Capability to manually isolate the line
		Hydraulic hammer		
		Improper maintenance		Check valve to prevent a large backflow through a line breach
		Instrument or instrument line failure		
		Material defect		Corrosion probes
		Thermal expansion with equipment blocked in		
		Low flow or no flow (see 3.2)		Periodic nondestructive inspection
		Reverse flow (see 3.3)		

Table 23: Reference HAZOP worksheet of the hexane tank based on (CCPS, 2001)

Color legend	Recognized	Indirectly recognized	Other conclusion

Pos.	Deviation	Cause	Consequence	Safeguard
4.1	High level	Flow from tank truck not discontinued before tank capacity has been reached	High pressure	Level indication with high-level alarm (audible in the control room)
		Inventory control error – truck arrives before needed		Hexane unloading procedures with a checklist that includes checking field reading of tank before unloading
4.2	Low level	Inventory control error – truck arrives too late	No safety consequences – potential process interruption if not refilled before the downstream feed tank is empty	-
		Low flow or no flow – line from the tank truck to hexane storage tank T-301 through hexane unloading pump		
4.3	High temperature	No credible causes identified	-	-
4.4	Low temperature	Low ambient temperature while there is water contamination in the tank	Possible freezing of accumulated water in the heel of the tank or the tank's drain line or instrument lines, resulting in fracture of the drain line and loss of containment	-
4.5	High pressure	High level	Release of hexane through the relief valve into the tank's dike	-
			Fire hazard affecting a large area if not contained by the dike	
			Loss of containment (if the overpressure cause exceeds the tank pressure rating)	
4.6	Low pressure	Tank blocked in before cooldown, following steam-out	Equipment damage resulting in a collapse of the tank under a vacuum	Standard procedures for steam-out of vessels
4.7	High concentration of contaminants (Other than composition)	Water not completely drained following a steam-out or washout	Possible freezing of accumulated water in the tank during a period of low ambient temperature	-
		High concentration of contaminants – line from the tank truck to hexane storage tank T-301 through hexane unloading pump		
4.8	Loss of containment (Elsewhere flow)	Corrosion	Release of hexane	Operation/maintenance response as required, including isolation if needed
		External fire		Capability to manually isolate the tank
		External impact		
		Gasket, packing or seal failure		Periodic non-destructive inspection per API recommended practice and ASME code
		Improper maintenance		
		Instrument or instrument line failure	Fire hazard affecting a large area particularly if the capacity of the dike is exceeded	Relief valve that discharges to the tank's dike
		Material defect		
		Sample station valve leaking		Dike sized for 1.5 times the capacity of the tank
		Vent or drain valve leaking		
		Low temperature		Emergency response procedures
		High pressure (if the overpressure cause exceeds the equipment pressure rating)		

Causes that originate in the tank truck, for example, only manifest themselves in the storage tank to form a scenario. A scenario with a moderate probability of occurrence is a substance confusion by lining up a tank with another substance, which in the worst case can lead to a "PotentialViolentReactionWithOxidizers" since hexane can react violently with oxidizers (cf. Table 21, Pos. 49), leading to a "PressureExceedingDesignPressure", and "Fracture". This consequence was not considered in the original worksheets (cf. Table 22, Pos. 3.8).

Concerning the pump, more operability issues that could lead to hazardous events have been recognized by the knowledge-based system. For instance, the dry-running or cavitation of the pump can lead to 'high vibration' which in turn can lead to a "FatigueFracture" over time (cf. Table 20, Pos. 15-17).

Furthermore, the consequence 'freezing and fracture of the drain line' was recognized within the original HAZOP worksheet with the deviation 'low temperature' (cf. Table 23, Pos. 4.4). This scenario was recognized with a combination of the deviations 'other than composition' and 'low temperature' (cf. Table 21, Pos. 36). Additionally, within the knowledge-based system, the boundary condition is checked whether the freezing point of water can fall below the lowest ambient temperature. Many causes are subsumed under the deviation 'loss of containment' without further description in the reference HAZOP worksheet. This applies to both the feed line and storage tank (cf. Table 22, Pos. 3.9; Table 23, Pos. 4.8). In the automatically generated worksheets, parts were assigned to other deviations. For example, "ExternalFire" was assigned to the deviation 'high temperature', while leakages, external impact, or maintenance error were assigned to the 'elsewhere flow' deviation.

These examples show that different terms can be used while similar conclusions can be drawn. This issue is also typical within conventional HAZOP studies because experts use different vocabulary and deviations depending on company-specific guidelines. The original HAZOP worksheet lists deviations, causes, and consequences separately, corresponding to the typical HAZOP table layout. Instead of considering scenario components implicitly within the textual description, they are explicitly considered as separate columns in the automatically generated results.

In the worst case, a release and a fire hazard are assumed within the original HAZOP (cf. Table 23, Pos. 4.5). The proposed method also considers the consequence "RiskOfExplosiveAtmosphere" as a potential consequence based on the hazard information 'may form an explosive mixture in air' in the SDS (cf. DHC, 2017). Typically, a rough consequence analysis is also conducted as part of HAZOP in the industrial environment to assess the risk of individual scenarios.

Besides company-specific guidelines and the type of industry, the extent of the safeguards depends on the risk category of the scenario, and the automatically identified proposals of safeguards take this dependency into account. Since no risk assessment is included within the original HAZOP, this dependency exists only implicitly through the experience of the HAZOP conductors, who assess the scenarios before safeguards are identified. Safeguards were not presented for all scenarios in the reference HAZOP tables, indicating the incompleteness of the original HAZOP table. Nevertheless, individual safeguards apply across scenarios, e.g., 'hexane unloading procedure with a checklist' or 'periodic non-destructive inspection' (cf. Table 23, Pos. 4.1).

Despite the partially different ways of using deviations and presenting scenarios, 15 of 19 safeguards of the reference HAZOP worksheets have been identified, as indicated by the colors in Table 22 and Table 23. In general, the developed knowledge-based system identified significantly more safeguards than listed in the reference worksheets. For some scenarios, other safeguards have been identified. A dike was not identified as a safeguard using the proposed method since this corresponds to a structural measure that must be provided for in the basic design of the tank and cannot easily be considered retrospectively (cf. Table 23, Pos. 4.8). The safeguard 'capability to manually isolate the tank' (cf. Table 22, Pos. 4.8) was not recognized since a shut-off valve at the inlet and outlet of the tank are part of the equipment entity. The safeguard 'local pressure gauge'

in the reference worksheet (Table 22, Pos. 3.7) is an operational measure. It does not constitute a safeguard without a further measure such as "DetermineSafetyRelatedOperatingInstructions" (cf. Table 20, Pos. 1).

Some of the described measures, such as 'standard procedure for steam-out of vessel' (cf. Table 23, Pos. 4.6), implicitly imply the installation of a "PressureVacuumReliefValve", which was recognized several times (cf. Table 21). Due to the flashpoint of -20 °C of hexane, a minimum distance between tanks is identified based on the recommendation in TRGS 509 (cf. "ConsiderMinimumTankDistance", Table 20, Pos. 28-31f.). Some safeguards identified by the knowledge-based system are more comprehensive than in the original HAZOP, which results from the performed risk assessment, e.g., a scenario ranked with risk category **B** must be reduced to category **C** using safeguards. A direct comparison of the safeguards is shown in Table 24. For some scenarios, a set of safeguards is identified, which are suggestions and where an experienced engineer must make a final selection depending on good engineering practice and company policies. For instance, in Table 21, Pos. 38-41 multiple safeguards are suggested to prevent severe corrosion of the storage tank. The selection depends on whether the product can be contaminated ("AddCorrosionInhibitor"), whether the material used can be easily replaced ("ConsiderMaterialSelection"), or whether a new coating is sufficient ("ProvideAntiCorrosionCoating").

In summary, all safety-relevant scenarios of the original HAZOP worksheets were identified with the help of the knowledge-based system. Also, more operational issues with safety-relevant potential have been recognized. For instance, the effects "PumpRunningDry" and "Cavitation" can lead to increased vibration with further consequences (cf. Table 20, Pos. 15-17). In addition, more in-depth scenarios such as the "GenerationOfElectrostaticCharge" and a substance mix-up with subsequent "PotentialViolentReactionWithOxidizers" could be identified.

Table 24: Comparison of the identified safeguards in case study I

Pos.	Original HAZOP		Knowledge-based System	
	Description	**Reference**	**Identified classes**	**Reference**
1	"Check valve to prevent a large backflow through a line breach"	Table 22, Pos. 3.3/3.9	"InstallCheckValve"	Table 20, Pos. 20
2	"Hexane unloading procedure"	Table 22, Pos. 3.8	"DetermineSafetyRelatedOperatingInstruction"	Table 20, Pos. 1
3	"Caps for flexible unloading line"	Table 22, Pos. 3.8	"UseSealingCaps"	Table 21, Pos. 40
4	"Hexane unloading procedure"	Table 22, Pos. 3.8	"PeriodicalSampleTaking" "ImplementQuickConnectSystem"	Table 21, Pos. 39 Table 21, Pos. 49
5	"Corrosion probes" "Periodic non-destructive inspection"	Table 22, Pos. 3.9	"PeriodicalSampleTaking" "RegularPlantInspection"	Table 21, Pos. 39 Table 20, Pos. 24-25
6	"Hexane unloading procedures with a checklist that includes checking field reading of tank before unloading"	Table 23, Pos. 4.1	"DetermineSafetyRelatedOperatingInstruction" "ImplementQuickConnectSystem"	Table 21, Pos. 49
7	"Level indication with high-level alarm (audible in the control room)"	Table 23, Pos. 4.1	"DefineMaximumFillLevel", "OverfillProtection" "InstallHighLevelAlarm"	Table 20, Pos. 24-25
8	"Standard procedure for steam-out of vessel"	Table 23, Pos. 4.6	"PressureVacuumReliefValve" "EmergencyPressureReliefValve"	Table 20, Pos. 30

6.2 Case study II: Water separator system

In the following case study, the feed section of an olefin dimerization unit is considered. The case study was chosen because it has already served as a benchmark within literature approaches, and thus results can be compared with each other (cf. Parmar and Lees, 1987; Kang et al., 2001). The considered plant section is shown in Figure 82. It consists of a transfer pump, half a mile piping, control valve, settling tank, and multiple shut-off and non-return valves. The contained hydrocarbon stream consists of a mixture of 50 mole % hexane and 50 mole % hexene with a small amount of suspended water fed into a reactor in which a dimerization reaction occurs. The feed section has the intended function of removing water from the mixture, which has an adverse effect on the dimerization reaction, which takes place downstream. Only the plant section up to the settling tank is considered in this case study.

6.2.1 Model representation of the water separator system

A section of the P&ID of the feed section is shown in Figure 82. The plant section must be transformed into an SDG model for the automatic analysis, shown in Figure 83. In the original description of the feed section, the feed pump is redundantly installed; only one pump is considered in this case study because one pump line is permanently shut off unless one pump fails, in which case the second pump is switched on. The equipment entities, *centrifugal pump*, *pipeline*, *control valve*, *settling tank*, and *nitrogen blanketing* are considered. Additionally, the beginning and end of the section are represented by the nodes *source* and *sink*. The settling tank is specified with a capacity of 50 m^3, while 25 m^3 of it is liquid. The volume flow of the feed pump is 25 m^3/h.

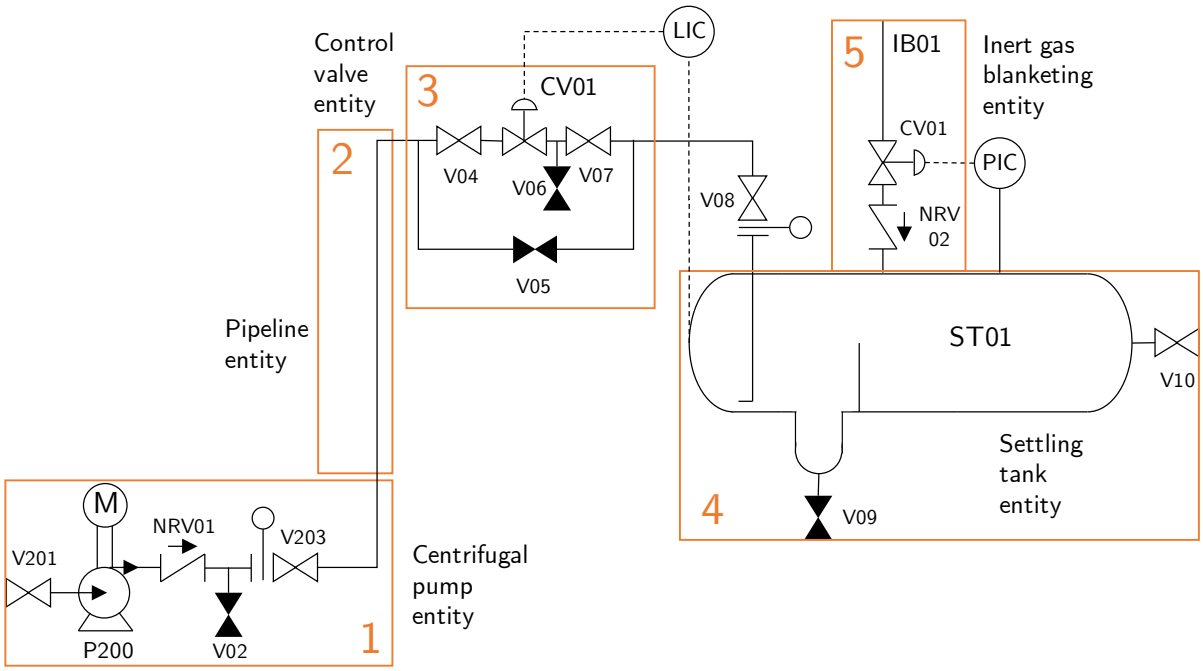


Figure 82: Feed section of the olefin dimerization plant (cf. Parmar and Lees, 1987)

Table 25: Function description matrix of the olefin feed section

Type	Pump	Connection pipe	Control valve	Inert gas blanketing	Settling tank
IntendedFunction	DeliverVolumeFlow	ModeIndependent	FlowControl	Inerting	Separating
Operation mode	NormalOperation	NormalOperation	NormalOperation	NormalOperation	NormalOperation

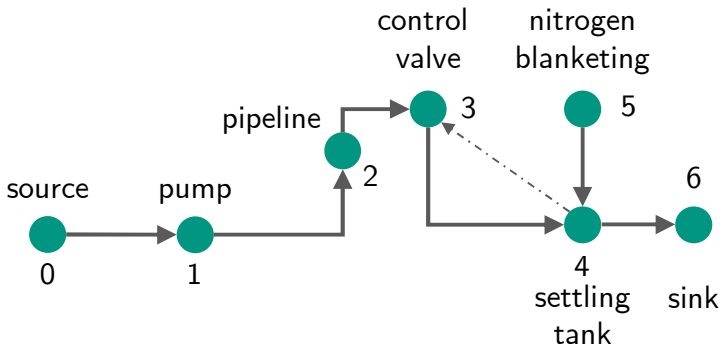


Figure 83: SDG representation of the feed section of olefin dimerization plant

Other fittings in the feed section include non-return valves, shut-off valves, and bypasses. A flow measurement is included in the feed line in the original illustration. Still, it is not considered here since the measured value is only read manually and is not a controlled variable.

The pressurized settling tank is overlaid with the inert gas nitrogen and has a working pressure of 15 psig. The flow control valve is connected to the settling tank via a level indicator and controller. The nitrogen blanketing unit is equipped with a pressure control valve connected to a pressure indicator controller of the settling tank. Table 25 shows the function description matrix, which describes the intended function of the involved equipment entities and the overall function of the node.

In the original description, the involved substance is described as an olefin consisting of 50 mole % hexane and 50 mole % hexene, contaminated by water. There is hexane, hexene, water mixture in the feed line to the settling tank; there is also nitrogen present in the settling tank, while there is only nitrogen in the inert gas blanketing unit. The hazard classes and further information on the substances involved are obtained based on the SDS (cf. Table 18; Table 26). Additional boundary conditions are:

- the plant section is located outside (**LocatedOutside**), which implies that there is always sufficient oxygen available (**SufficientOxygenAvailable**),
- that contamination or impurities may be introduced into the subsystem (**IntroductionOfImpurities**).
- Since employees can be present in the near field of the plant section, the boundary condition **PersonalPresentInTheNearfield** is set,
- it must be assumed that an external fire can occur (**ExternalFirePossible**),
- The feed line (35 psig) and the settling tank (15 psig) are under pressure, and it is assumed that no air can enter the system from the outside. It is assumed that air can be sucked in based on upstream conditions ("IntroductionOfAir").

Within this plant section, all boundary conditions apply equally to all equipment entities. The identification of suitable safeguards is part of the HAZOP study. Accordingly, existing safety measures, e.g., a safety valve on the settling tank, are not considered during system modeling and must be identified by the knowledge-based system. The developed equipment entity library was used to model the equipment entities. The formal input data object is shown in Appendix Listing W-2 and Appendix Listing W-3.

Table 26: Excerpt from the Safety Data Sheet: Hexene

Hexene			
Lower explosion limit	1.2 vol%	**Upper explosion limit**	6.9 vol%
Odor	Characteristic	**Vapor pressure**	24.8 kPa @ 25 °C
pH-value	Not determined	**Density**	0.678 g/cm^3
Melting/Freezing point	-140 °C	**Viscosity**	0.34mm^2/s @ 40 °C
Boiling point/Boiling range	62-65 °C	**Auto/Self-Ignition temperature**	265 °C
Flash point (closed cup)	-26 °C	**Decomposition temperature**	Not determined

Hazard class	**Category**
Flammable liquid	Cat. 2
Specific target organ toxicity - single exposure	Cat. 3
Aspiration hazard	Cat. 1

Stability and reactivity information
Hazardous polymerization may occur
Incompatible with oxidizing agents, acids, peroxides

6.2.2 Automatically generated HAZOP results regarding the olefin feed section

The automatically created HAZOP worksheets of the feed section are shown in Table 27 and Table 28. Overall, 63 potential scenarios were identified. Regarding the centrifugal pump in the feed section, safety-relevant scenarios, as well as operability issues, were identified. Concerning the 'no flow' deviation, scenarios with the effects "FluidCirculatesInsidePump", "Overheating", "PumpRunningDry", "PumpDeliversNoLiquid" were identified, which have the causes "ClosedInletValve", "DeadHeadingOfPump" and "MissingImpeller" (cf. Table 27, Pos. 1-4). Due to the properties of the hydrocarbon mixture, the pump may pump against a line containing polymerized olefin, which may result in 'overheating' of the pump with the consequence 'pump breakdown' (cf. Table 27, Pos. 12).

Another operability scenario describes an "ImpellerFault", which leads to "IncreasedWear" (cf. Table 27, Pos. 5). This scenario was rated as uncritical with the consequence "PumpBreakdown", and "ProductionDowntime" and the risk ranking **D**. Operation below the minimum flow rate can lead to overheating within a centrifugal pump since the heat introduced by the drive is not sufficiently removed from the fluid being pumped, which in the worst case leads to a "PumpBreakdown" (cf. Table 27, Pos. 6). Another typical pump operability scenario involves an "InsufficientNpsh" (insufficient net positive suction head), which can lead to "Cavitation" (cf. Table 27, Pos. 7). Typically, operability issues are classified with risk category **D**.

In the worst case, the operability issues within the pump lead to a "PumpBreakdown", which can result in a "ProductionDowntime". Operability issues can lead to safety-relevant scenarios. For example, the effects "PumpRunningDry" or "Cavitation" can lead to the deviation **HighVibration,** which can, in turn, lead to a "FatigueFracture" over time (cf. Table 27, Pos. 14-15). Since these scenarios are ranked with the risk category **B**, the safeguard "InstallVibrationDampener" was identified. Relating to the pump scenarios 8 and 9 in Table 27, describe a "PoolFormation", which is expressed by the deviation **ElsewhereFlow** and can have the causes "PumpSealFailure" and "DrainValveInadvertentlyOpened". While the pump seal failure can be caused by mechanical wear in the form of the underlying cause "WearDown", the unintentional opening of the drain valve can be attributed to an "OperationalError". As a safeguard for the pump seal failure, "ElaborationOfMaintenanceConcept" was identified, while an inadvertently opening of a valve can be safeguarded with a "ValveLockedClosed". The safeguard

"RegularPlantInspection" generally applies to scenarios involving loss of containment, as this can be detected at an early stage so that further measures can be taken.

Within the feed line to the settling tank, a "WaterHammer" was identified as the cause of excessive pressure with a "RapidlyClosingValve" as the underlying cause (cf. Table 27, Pos. 24). Since the case study involves a centrifugal pump, a sudden stop of the pump due to the inertia of the impeller will not necessarily result in a water hammer. For a positive displacement pump, this would also be a plausible scenario. Also, the **UnderlyingCause** "BlockedPipingAndHeatInput" can lead to a "ThermalExpansion" inside the pipeline, which can cause a "PressureExceeding-DesignPressure" with a subsequent "Fracture" (cf. Table 27, Pos. 23). Also, a "LossOfLeakTightness" or "DefectiveSeal" may lead to an "ExternalLeakage" within the pipeline, resulting in a "PoolFormation" with the consequence "LossOfPrimaryContainment" (cf. Table 27, Pos. 21-22). Additionally, one scenario was identified within the propagation mode involving the **HighFlow** deviation within the pipeline.

A "PumpIncorrectlySet" or "WrongImpeller" can lead to a 'high flow' with the effect "GenerationOfElectrostaticCharge" and the consequence "EmergenceOfIgnitionSource", which can have subsequent severe consequences due to the flammability of the mixture and is accordingly classified as risk category **B** (cf. Table 27, Pos. 27-28). The safeguards "LimitationOfTheFlowVelocity" and "ProvideSufficientEarthing" have been identified to prevent this scenario.

When pumping liquid against a closed gate valve, e.g., the control valve, an impermissible pressure build-up may occur in the piping system. This scenario of an unintentionally closed control valve with the cause "ValveClosedPressureBuildUpInPiping" and the underlying causes "MalfunctionControlAir", and "OperationalError" was identified (Table 27, Pos. 29-30).

For safe operation, the pressure class of the pipeline must be above the shut-off head of the pump plus the suction pressure. Accordingly, "CheckPressureClassPiping" and "RegularPlantInspection" have been identified as the safeguard for this scenario (cf. Table 27, Pos. 29-30).

Further scenarios in the settling tank arise in interaction with the control valve. The 'high flow' deviation in the settling tank can be attributed to the control valve with the causes "BypassOpened" or "ValveWronglyOpened". The identified underlying causes of it are "MalfunctionControlAir", "ControlValveFailsOpen", "LevelIndicatorControllerFailure", or "OperationalError" (cf. Table 28, Pos. 56-60). A 'high flow' into the settling tank can lead to the effect "PoorSeparation".

The cause "OtherSubstanceFromUpstream" already occurs in the source and propagate through the rest of the plant section (cf. Table 28, Pos. 45). Thus, it has its origin outside the system boundary. Additionally, the deviation 'other than composition' can have its origin in "ReducedDwellTime", which leads to the effect of "PoorSeparation" (cf. Table 28, Pos. 31). These scenarios are not directly relevant to safety for this plant section but can lead to process problems in the downstream reactor.

The 'low pressure' scenario with the effect "LossOfMechanicalIntegrity" can have the underlying cause "PressureIndicatorControllerFailure" and the cause "IncorrectPressureAdjustment" within the settling tank because of the nitrogen blanketing. Without a pressure adjustment by the inert gas blanketing unit, the negative pressure in the settling tank with a subsequent "LossOfMechanicalIntegrity" may occur (cf. Table 28, Pos. 34). This scenario is particularly plausible if the tank has a drain valve, leading to the formation of underpressure during excessive draining.

Besides the 'low pressure' scenario, five 'high pressure' scenarios are identified within the settling tank. These may be due to overfilling of the tank (cf. Table 28, Pos. 40) "AbnormalEvaporation" (cf. Table 28, Pos. 43-44). Since the likelihood of "AbnormalEvaporation" in conjunction with a fracture within a settling tank (pressure vessel) caused by "SolarRadiation" (likelihood: **Unlikely**) is lower compared to "ExternalFire", the risk ranking differs for these scenarios involving the risk categories **B** and **B**/**C** (cf. Table 28, Pos. 43-44).

Table 27: Automatically generated extended HAZOP worksheets of the olefin feed section I

Pos.	Equipment entity	Id.	Intention / mode	Substance	Deviation	Underlying cause	Cause	Effect	Consequence	Safeguard	Propagated	Risk
1	PumpEntity	P200	DeliverConstantVolumeFlow	Olefin&Water	NoFlow	OperationalError	ClosedInletValve	PumpRunningDry	PumpBreakdown, ProductionDowntime	DryRunProtection, ValveLockedOpen	False	D
2						ValveFailure					False	D
3						OperationalError	DeadHeadingOfPump	FluidCirculatesInsidePump, Overheating	DangerOfBleve	ImplementNoFlowAlarm, OverFlowValveAndKickBackLine, OverflowValve, ThermalProtectionTrip, ValveLockedOpen	False	B
4						MaintenanceError	MissingImpeller	PumpDeliversNoLiquid	ProductionDowntime	ImplementNoFlowWarning	False	D
5					LowFlow	-	ImpellerFault	IncreasedWear	PumpBreakdown, ProductionDowntime	LowFlowProtectionTrip	False	D
6						-	OperationBelowMinimumFlowRate			LowFlowProtectionTrip, StandardOperationProcedure	False	D
7					LowPressure	-	InsufficientNPSH	Cavitation		IncreaseSuctionPressure, InstallPumpInducer	False	D
8					ElsewhereFlow	WearDown	PumpSealFailure	PoolFormation	LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	ElaborationOfMaintenanceConcept, RegularPlantInspection	False	B
9						OperationalError	DrainValveInadvertentlyOpened			RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B
10					HighTemperature	OperationalError	P200: ClosedInletValve -> PumpRunningDry -> HighTemperature (Liquid)	Overheating	PumpBreakdown, ProductionDowntime	OverFlowValveAndKickBackLine, ThermalProtectionTrip	False	D
11						ValveFailure					False	D
12						-	P200: PumpingAgainstPolymerizedLine -> FluidCirculatesInsidePump -> HighTemperature (Liquid)				False	D
13						OperationalError	P200: DeadHeadingOfPump -> FluidCirculatesInsidePump -> HighTemperature (Liquid)				False	D
14					HighVibration	OperationalError	P200: ClosedInletValve -> PumpRunningDry -> HighVibration (Liquid)	FatigueFracture	LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	ImplementVibrationDampener, RegularPlantInspection	False	B
15						ValveFailure					False	B
16						-	P200: InsufficientNPSH -> Cavitation -> HighVibration (Liquid)				False	B
17					ElsewhereFlow	OperationalError	LeakingDrainValve	PoolFormation		RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B, C
18						MaintenanceError					False	B, C
19						LossOfLeakTightness				ElaborationOfMaintenanceConcept, RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B, C
20						DefectiveSeal					False	B, C
21							ExternalLeakage				False	B
22						LossOfLeakTightness					False	B
23	ConnectionPipeEntity	PIP1	ModeIndependent		HighPressure	BlockedPipingAnd-HeatInput	ThermalExpansion	Fracture, PressureExceedingDesignPressure		CheckPressureClassPiping, InstallThermalExpansionReliefValve, RegularPlantInspection	False	B, C
24						RapidlyClosingValve	WaterHammer			CheckPressureClassPiping, IncreaseClosingTimeOfValve, PulsationDampener, RegularPlantInspection, SurgeReliefValve, SwingCheckValve	False	B
25					ElsewhereFlow	LossOfLeakTightness	ExternalLeakage	PoolFormation		ElaborationOfMaintenanceConcept, InstallCheckValve, RegularPlantInspection	False	B
26					HighVibration	RapidlyClosingValve	PIP1: WaterHammer -> HighVibration (Liquid)	FatigueFracture		RegularPlantInspection	False	B
27					HighFlow	OperationalError	P200: PumpIncorrectlySet -> HighFlow (Liquid)	GenerationOfElectrostaticCharge	EmergenceOfIgnitionSource	ConsiderMaterialSelection, LimitationOfTheFlowVelocity, ProvideGroundingOfPlant	True	B
28							P200: WrongImpeller -> HighFlow (Liquid)				True	B
29	ValveEntity	CV01	FlowControl		HighPressure		ValveClosedPressureBuildUpInPiping	Fracture, PressureExceedingDesignPressure	LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	CheckPressureClassPiping, RegularPlantInspection	False	B
30						MalfunctionControlAir					False	B

For polymerization of hexene to occur, a radical initiator or a strong heat supply is required. Since the external addition of a radical initiator for this plant section can be reasonably excluded, a strong heat input remains, caused by an external fire. The scenario of an "UnintendedExothermicPolymerization" leads to a pressure increase and a "Fracture" with a subsequent "LossOfPrimaryContainment" (cf. Table 28, Pos. 41-42). This scenario results in the risk category **B** and **B/C** with the following safeguards "PressureReliefValve", and "RegularPlantInspection". The flare system is identified because of the hazardous properties of the mixture.

In addition, operational problems related to the inertization unit without protection can lead to a "high pressure" scenario. An "IncreasedInletPressure" within the inert gas blanketing unit, increased inlet pressure, or an excessive inflow can lead to overpressure (see Table 28, Pos. 63). The overpressure or underpressure scenarios can result in a "LossOfPrimaryContainment", which can lead to further consequences due to the hazardous properties of olefin.

Due to the stability and reactivity information **PolymerizesExothermicallyWhenExposedToHeat** of hexene, a scenario has been identified in which polymerization in the feed line blocks the flow into the settling tank the tank runs empty (cf. Table 28, Pos. 51). A risk class **D** is identified for this scenario because a polymerization only due to solar radiation is considered unlikely.

Additionally, an "InsufficientInertization" by nitrogen can lead to the consequence "RiskOfExplosiveAtmosphere" due to the hazardous attributes of the hexane/hexene (olefin) mixture (cf. Table 28, Pos. 61-62). The likelihood category of a "UtilityFailure" is **Likely** (cf. Appendix Listing O-2 and due to the severity category **Severe,** the overall scenario is ranked with the risk category **B**. The safeguard "InstallNoFlowAlarm" and "FailsafeFeedStop" are identified to compensate for the failure of inertization. To prevent the settling tank from being flooded with nitrogen, the safeguard "InstallPressureLimitationValve" is proposed (cf. Table 28, Pos. 63).

6.2.3 Propagation based analysis of the olefin feed section

Within the propagation mode, the layout of the considered plant section is considered. Since two lines lead into the settling tank, the plant section is a 'junction system' (cf. Section 5.14.2). Thus, the SDG representation of the plant is automatically decomposed into different lines that are evaluated separately. The algorithm developed for this has already been described in Section 5.14.2. The SDG representation of the original plant section and the decomposed and rearranged graphs are shown in Figure 84. First, the system is considered from the source to the settling tank, then the nodes between nitrogen blanketing and the settling tank are considered, and then the nodes settling tank and sink are considered. The lines represent how the equipment entities are analyzed, and events are propagated through the lines. The event is further propagated if propagated events have no immediate effect or consequence in the current equipment entity. The propagation mode of the inference algorithm provides explanations for scenarios as described in Section 6.2.2.).

Scenarios identified within the propagation mode are marked with a *true* in the propagated column of the automatically generated HAZOP worksheets (cf. Table 27 and Table 28). The further the algorithm progresses in the plant section, the more propagation scenarios occur. 19 of 63 overall scenarios have been identified within the propagation mode. Most propagating scenarios refer to the settling tank since it is the last element with the system boundary, and both lines (feed section, nitrogen blanketing) end in it. Within the last node of the analyzed system, propagating events are not considered further since they would only become active outside the system boundary under consideration.

Table 28: Automatically generated extended HAZOP worksheets of the olefin feed section II

Pos.	Equipment entity	Id.	Intention / mode	Substance	Deviation	Underlying cause	Cause	Effect	Consequence	Safeguard	Propagated	Risk
31			Separating		OtherThanComposition	-	ReducedDwellTime	PoorSeparation	PoorProductQuality	ImplementFrequentDrainingOff	False	D
32					LowTemperature, OtherThanComposition	-	ContaminationByWaterAnd-TemperatureFallsBelowFreezingPoint	DrainlineFracture	LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	InstallHeatTracingSystem, RegularPlantInspection	False	B, C
33					HighPressure	ExternalFire	ThermalExpansion	Fracture, PressureExceedingDesignPressure	LossOfPrimaryContainment, FireHazard	FlareSystem, PressureReliefValve, RegularPlantInspection	False	B
34					LowPressure	PressureIndicator-ControllerFailure	IncorrectPressureAdjustment	LossOfMechanicalIntegrity		RegularPlantInspection, VacuumProofDesign	False	B
35					ElsewhereFlow	OperationalError	LeakingDrainValve	PoolFormation		RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B, C
36						MaintenanceError				RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B, C
37						LossOfLeakTightness			LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	ElaborationOfMaintenanceConcept, RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B, C
38						OperationalError	DrainValveInadvertentlyOpened			RegularPlantInspection, StandardOperationProcedure, ValveLockedClosed	False	B
39						LossOfLeakTightness	ExternalLeakage			ElaborationOfMaintenanceConcept, RegularPlantInspection	False	B
40			Mode-Independent		HighPressure	FailureControlLoop	ST01: IncorrectIndicationOfFillingLevel -> Overfilling -> HighPressure (Multiphase)	Fracture, PressureExceedingDesignPressure		FlareSystem, PressureReliefValve, RegularPlantInspection	False	B
41				Olefin&Water&-Nitrogen		SolarRadiation	ST01: AbnormalHeatInput -> UnintendedExothermicPolymerization -> HighPressure (Multiphase)			FlareSystem, InstallHighTemperatureAlarm, PressureReliefValve, RegularPlantInspection	False	B, C
42	SettlingTankEntity	ST01				ExternalFire	ST01: AbnormalHeatInput -> UnintendedExothermicPolymerization -> HighPressure (Multiphase)		LossOfPrimaryContainment, FireHazard		False	B
43						SolarRadiation	ST01: AbnormalHeatInput -> AbnormalEvaporation -> HighPressure (Multiphase)		LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	FlareSystem, PressureReliefValve, RegularPlantInspection	False	B, C
44						ExternalFire			LossOfPrimaryContainment, FireHazard		False	B
45					OtherThanComposition	-	Source: OtherSubstanceFromUpstream -> OtherThanComposition (Liquid)	PoorSeparation	PoorProductQuality	ImplementFrequentDrainingOff	True	D
46					NoFlow	OperationalError	P200: ClosedInletValve -> NoFlow (Liquid)	EmptyingOfContainer	ProductionDowntime	ImplementLowLevelWarning, ValveLockedOpen	True	D
47						MalfunctionControlAir	CV01: ValveWronglyClosed -> NoFlow (Liquid)			ImplementLowLevelWarning, UseNormallyOpenValve	True	C
48						OperationalError				ImplementLowLevelWarning, StandardOperationProcedure	True	C
49						ControlValveFailsClosed				ElaborationOfMaintenanceConcept, ImplementLowLevelWarning	True	C
50						LevelIndicatorControllerFailure				ImplementLowLevelWarning, IncreaseSafetyIntegrityLevel	True	C
51						SolarRadiation	P1P1: AbnormalHeatInput -> UnintendedExothermicPolymerization -> NoFlow (Liquid)	EmptyingOfContainer		AvoidingBlockedLiquids, ImplementLowLevelWarning	True	D
52						MaintenanceError	P200: MissingImpeller -> NoFlow (Liquid)			ImplementLowLevelWarning, StandardOperationProcedure	True	D
53						LevelIndicatorControllerFailure	CV01: IncorrectSetPointControlValve -> NoFlow (Liquid)			ImplementLowLevelWarning, IncreaseSafetyIntegrityLevel	True	C
54						OperationalError				ImplementLowLevelWarning, StandardOperationProcedure	True	C
55							CV01: ValveIntactUnintentionallyClosed -> NoFlow (Liquid)				True	D
56			Separating		HighFlow	OperationalError	CV01: ValveWronglyOpened -> HighFlow (Liquid)	PoorSeparation	PoorProductQuality	ImplementFrequentDrainingOff	True	D
57						ControlValveFailsOpen					True	D
58						LevelIndicatorControllerFailure				ImplementFrequentDrainingOff, IncreaseSafetyIntegrityLevel	True	C, D
59						MalfunctionControlAir				ImplementFrequentDrainingOff	True	C
60						OperationalError	CV01: BypassOpened -> HighFlow (Liquid)			ImplementFrequentDrainingOff, ValveLockedClosed	True	C
61			Mode-Independent	Nitrogen	NoFlow	UtilityFailure	NoInertgasSupply	InsufficientInertization	RiskOfExplosiveAtmosphere	FailsafeFeedStop, FlameArrestor, ImplementNoFlowAlarm	False	B
62				Nitrogen	OtherThanComposition	-	OtherSubstanceFromUpstream			FailsafeFeedStop, FlameArrestor	False	B
63				Olefin&Water&-Nitrogen	HighPressure	MalfunctionPressureController	IB01: IncreasedInletPressure -> HighPressure (Gaseous)	Fracture, PressureExceedingDesignPressure	LossOfPrimaryContainment, RiskOfExplosiveAtmosphere, FireHazard	InstallPressureLimitationValve, PressureReliefValve, RegularPlantInspection	True	B

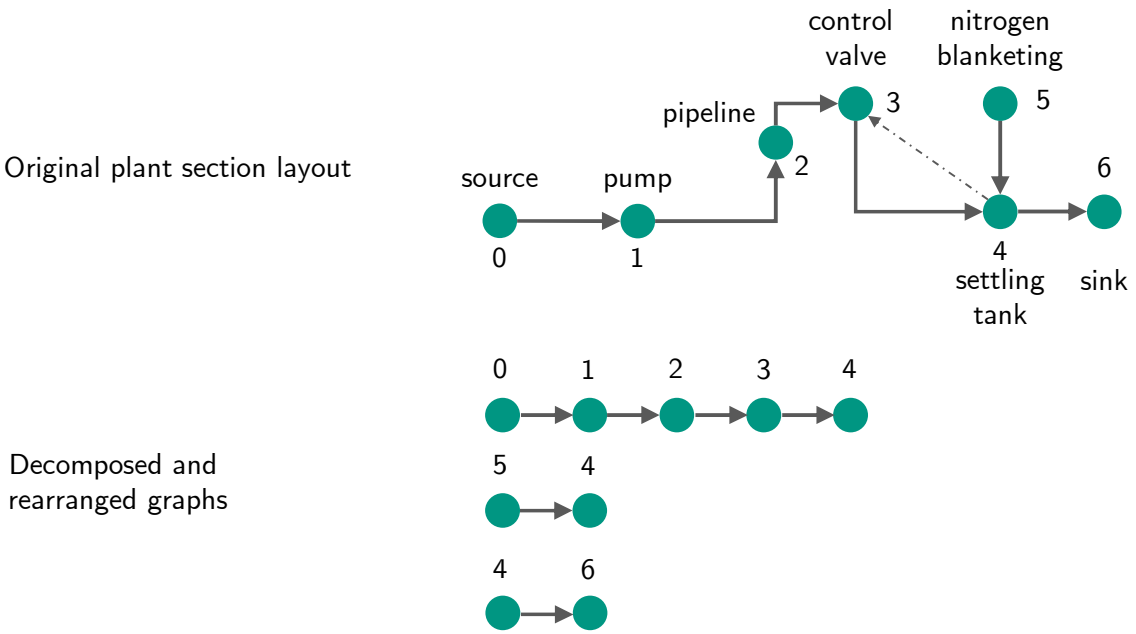


Figure 84: SDG representation and decomposition of the feed section

6.2.4 Original HAZOP results of the olefin feed section

Within the AutoHAZID approach, the olefin feed section has already been used as a benchmark. Parmar and Lees (1987) compared the original HAZOP results with the HAZID approach. The original HAZOP results are shown in Mannan (2012), pp. 8/39. The results are shown in Table 29 and Table 30. In the HAZID results, identifying causes and consequences is conducted by differentiating between the equipment, whereas in the original HAZOP, only deviations were distinguished. The typical phenomenon that different HAZOP practitioners use different deviations can also be observed here.

Regarding the deviation 'other than composition', the deviations 'high concentration of water/alkanes/alkenes/organic acids' are listed (cf. Table 29, Table 30). Although a settling tank is involved, the deviations 'high level' and 'low level' are not considered. A too high or too low level of the settling tank is considered within the HAZID results via the deviations 'more flow' and 'less flow'. In the HAZID approach, 52 of the 98 causes are related to the pump, or the control valve, while 22 of the 36 consequences are related to the settling tank. Within the original HAZOP, a polymerization in a downstream section is assumed (cf. Table 29, Pos. 1). Also, in the HAZID results concerning the *source*, the causes are often simply the current deviation, which means that the deviation is assumed and its cause lies upstream, i.e., outside the system boundaries. For instance, the 'no flow' deviation has the cause 'q1 none' regarding the source (cf. Table 29, Pos. 1). Besides causes and consequences, no safeguards have been identified within the original and the HAZID approach. Cross-references have not been used within the original results. As in the previous case study, a color legend was used in the original HAZOP results table (cf. Table 29, Table 30) to indicate whether the developed knowledge-based system identified scenarios and to what degree (recognized, indirectly recognized, other conclusions).

6.2.5 Differences in content and quality

From a qualitative point of view, all safety-relevant scenarios from the literature and the AutoHAZID approach were identified, which corresponds to the attribute *completeness* (cf. Chapter 2 (9)). Safeguards and the risk assessment of the scenarios have not been covered within the literature and the AutoHAZID results. There are also differences in quality concerning the

characteristics, *accuracy,* and *relevance* of the identified HAZOP-specific results compared to the results of the developed method (Chapter 2 (9)).

The comparison of HAZOP results requires the analysis and interpretation of scenarios. Within the HAZID approach, the identified causes and consequences are categorized into the individual equipment, such as source, pipe, pump, non-return valve, meter, pipeline control valve, valve, settler. Not for all deviations, causes and consequences were identified for each equipment entity. This is reasonable because deviations can change into other deviations, or deviations can only be propagated into the downstream equipment without directly leading to scenarios in each equipment. Other causal chains may arise, and scenarios may move towards other deviations due to the different categorization of events. Accordingly, the results must be compared entirely, and individual scenarios, including specific deviations, cannot always be compared directly.

In the (Auto)HAZID approach, the equipment 'source' is given for all deviations. The 'source' is the first 'equipment' within a process plant and forms the interface to other equipment entities. Regarding the causes for the deviation within the source, the deviations themselves are often listed. For example, in Table 29, Pos. 1 as the cause for the 'no flow' deviation 'q1 none' is listed, which means that 'no flow' is the cause of 'no flow'. This representation does not meet the quality characteristic *relevance* (Chapter 2). The proposed method does not consider these 'trivial' causes since there is an unambiguous semantic difference between a cause and a deviation. In the HAZID approach, consequences are given as deviations, e.g., the 'no flow' deviation leads to an 'l5 none', which means no level in the settling tank. Due to the disjointness between the concepts deviation and consequence, this is impossible within the own approach. Nevertheless, this scenario is recognized in various scenarios (cf. Table 28, Pos. 45-55).

The use of a deviation as a consequence weakens a clear representation of scenarios and thus does not fulfill the quality criterion *accuracy* (cf. Table 30, Pos. 18 & 26f.).

The process equipment 'meter' was not considered within the proposed approach because it is a manual instrument without control influence on the operation of the plant section. The shown scenarios (cf. Table 29, Pos. 9) refer to the bypass or leakage and are only indirectly related to the "meter". Concerning the pump, the HAZID approach compared to the own approach a comparable number of causes identified. The effects and consequences of the own approach are more extensive than the HAZID results. For example, the consequence 'fluid overheats' was identified within HAZID (cf. Table 29, Pos. 2), whereby the following effects were identified with the help of the proposed method (cf. Table 27):

- "FluidCirculatesInsidePump",
- "PumpRunningDry",
- "PumpDeliversNoLiquid",
- "IncreasedWear",
- "Overheating",
- "Cavitation",
- "PoolFormation",
- "FatigueFracture",

and the consequences

- "PumpBreakdown",
- "ProductionDowntime",
- "LossOfPrimaryContainment".

Pump-related phenomena were resolved more finely since operability issues can also result in safety-relevant scenarios. For instance, cavitation can lead to vibrations, which can cause fatigue fracture. Also, scenarios are constructed differently, e.g., in the HAZID approach, cavitation is a cause of the 'low flow' deviation (cf. Table 29, Pos. 3), while in the proposed approach, it is an

effect of the deviation 'low pressure' (cf. Table 27, Pos. 7). Furthermore, in the HAZID approach, the causes for the deviations 'no flow' and 'less flow' are largely the same for pump-specific causes (cf. Table 29, Pos. 3 & 6).

Scenarios that could lead to a 'low pressure' or 'high pressure' scenario due to an error in nitrogen blanketing were not detected within the HAZOP approach (cf. Table 30). With the developed knowledge-based system, the 'low pressure' scenario involving a "PressureIndicatorControllerFailure", "IncorrectPressureAdjustment", and "LossOfMechanicalIntegrity" were detected. In addition, the flow-related 'high pressure' scenarios with the causes "IncorrectIndicationOfFillingLevel", and "MalfunctionPressureController" were identified (cf. Table 28, Pos. 40 & 63).

In the results of AutoHAZID, two particular scenarios are shown in Table 30, Pos. 18-19. While the general cause 'impurity in the source' was identified by the developed knowledge-based system (cf. Table 28, Pos. 45), the specification 'low alkanes/alkenes, organic acids' was not recognized. These particular descriptions require further information regarding upstream equipment and the process, which is outside the system boundary of the plant section under consideration. It cannot be generally assumed that contamination by organic acids or low alkanes/alkenes can occur for any system. Such specific statements represent special rules that cannot be easily abstracted and generalized. Usually, such specific correlations are based on the process understanding and experience of the HAZOP practitioners.

Consistent with the findings, the release of a flammable substance or a polymerization is assumed as the worst-case within the AutoHAZID approach. (cf. Table 29, Pos. 33-40). Additionally, due to the properties of the hexene/hexane mixture, there is a risk of forming an (external) explosive atmosphere in case of release, which has been considered in Table 28 but has not been considered in the reference worksheets. Hence, the degree of *completeness* concerning safety-relevant scenarios is higher within the automatically created worksheets.

Since only the original HAZOP worksheets specify safeguards, only these can be compared with the automatically identified safeguards by the knowledge-based system and not with AutoHAZID. A comparison of the recognized safeguards is shown in Table 31. Textual descriptions are used in the original HAZOP, while the knowledge-based system uses concrete class names. In general, the knowledge-based system identified all safety-related safeguards of the original HAZOP. Individual descriptions of the original HAZOP consist of a series of measures (cf. Table 30, Pos. 18). A total of 19 of 24 of the proposed safeguards were identified automatically (cf. Table 29, Table 30). In addition, additional safeguards were identified. In the automatically generated worksheets, safeguards sometimes occur several times and only refer to one scenario. For example, the safeguard "CheckPressureClassPiping" was detected for several scenarios (cf. Table 27, Pos. 23-24, 30). Also, the safeguard "RegularPlantInspection" is not scenario-specific but can be assigned to several scenarios in the automatically generated worksheets (cf. Table 28, Pos. 8-9, 14-22).

In some cases, specific safeguards of the original HAZOP were not recognized, such as:

- 'Extend pump suction line to 12 inches',
- 'Install a PG upstream of LCV and an independent PG on settling tank'.

These safeguards are very specific and cannot be directly abstracted or transferred in contrast to general measures. For example, the safeguard 'Install a PG upstream of LCV and an independent PG on settling tank' represents a specific measure that does not constitute a proper safety measure without further instructions, such as periodic differential pressure checks or acoustic alarms.

Table 29: Reference HAZOP worksheet of the olefin feed section I (cf. Parmar and Lees, 1987)

Color legend | Recognized | Indirectly recognized | Other conclusion

Pos.	Deviation	Causes			Consequences			Safeguards (actions)
		Original HAZOP	Parmar and Lees		Original HAZOP	Parmar and Lees		Original HAZOP
1	No flow	No hydrocarbon at intermediate storage	source	q1 none; source empty	Loss of feed to reaction section and reduced output. Polymer formed in heat exchanger under no flow conditions	settler	q1 none; outflow none; l5 none	Ensure good communications with operator; install low level alarm on settler
2		J1 pump fails (motor fault, loss of drive, impeller corroded away, etc.)	pipe	blockage; leak (major)	J1 pump overheats	pump	fluid overheats	Install kick-back on pumps; check design of pump strainers
3		Line blockage, isolation valve closed in error or LCV fails shut	pump	loss of NPSH; rotation fault; impeller fault; cavitation; low b.pt material; gassy material; delivery shut in; no change to standby; standby fail on demand; maloperation of valves; blockage; leak (major)		pump		
4		Line fracture	non return valve	valve stuck closed	Hydrocarbon discharged into area adjacent to public highway	settler		Institute regular patrolling and inspection of transfer line
5			pipeline	blockage; leak (major)				
6			control valve	loop fails closed; sensor fails closed; controller fails closed; control valve fails closed; set point moved closed; isolation valve closed				
7			valve	valve moved closed				
8	More flow	LCV fails open or LCV bypass open in error	source	q1 high	Settling tank overfills; Incomplete separation of water phase in tank, leading to problems in reaction section	settler	q1 high; poor settling; i2 high; l5 high; liquid enters vent	Install high level alarm on settler and check sizing relief opposite liquid overfilling
9			meter	bypass fails; bypass moved open		meter	inaccurate measurement	Institute locking off procedure for bypass
10			control valve	loop fails open; sensor fails open; controller fails open; control valve fails open; set point moved open; bypass directed open		pipe		Extend pump suction line to 12 inch
11	Less flow	Leaking flange or valved stub not blanked and leaking	source	pi pl q1 low; blockage; leak (major)	Material loss adjacent to public highway	meter	inaccurate measurement	Institute regular patrolling and inspection of transfer line
12			pump	loss of NPSH; rotation fault; impeller fault; cavitation; low b. pt. material; gassy material; delivery part shut; blockage; leak (major)		settler	q1 low; outflow(s) low; l5 low	
13			non return valve	valve fails part closed				
14			pipeline	blockage; leak (major)				
15			control valve	loop fails part closed; sensor fails part closed; controller fails part closed; control valve fails part closed; set point moved part closed; isolation valve part closed				
16			valve	valve partly closed				
17	High water concentration in stream	High water level in intermediate storage tank	source	impurity (water) in source	Water sump fills up more quickly. Increased chance of water phase passing to reaction section	settler	poor separation of water; water sump overfills	Arrange for frequent draining off of water; install high interface level alarm on sump

Table 30: Reference HAZOP worksheet of the olefin feed section II (cf. Parmar and Lees, 1987)

Color legend | Recognized | Indirectly recognized | Other conclusion

Pos.	Deviation	Causes			Consequences			Safeguards (actions)
		Original HAZOP	Parmar and Lees		Original HAZOP	Parmar and Lees		Original HAZOP
18	High concentration of lower alkanes or alkenes in stream	Disturbance on distillation columns upstream of intermediate storage	source	impurity (low alkanes or alkenes) in source	High system pressure	pipe/ settler	pressure high	Check that design of settling tank and associated pipework, including relief valve sizing, will cope with sudden inflow
19	Organic acids present		source	impurity (organic acids) in source	Increased rate of corrosion of tank base, sump and drain line	pipe/ settler	corrosion	Check suitability of materials of construction
20	More pressure	Isolation valve closed in error or LCV closes, with J1 pump running	source	p1 high	Transfer line subjected to full pump delivery or surge pressure	pipe	leak (major); pipe rupture; flange leak	Check line FQ and flange ratings, and reduce stroking speed of LCV if necessary
21			pipe	blockage; heat to blocked-in fluid; solar heat; fire		settler	p1 high; overpressure; leak (major)	
22			pump	delivery shut in; blockage; heat to blocked-in fluid; solar heat; fire	Line fracture or flange leak	settler		Install a PG upstream of LCV and an independent PG on settling tank
23		Thermal expansion in an isolated valved section due to fire or strong sunlight	pipeline	blockage; hammer/surge; heat to blocked-in fluid; solar heat; fire				Install thermal expansion relief on valve section
24			control valve	loop fails closed; sensor fails closed; controller fails closed; control valve fails closed; set point moved closed; isolation valve closed				
25			valve	valve moved closed				
26	Less pressure		source	p1 low; source empty		settler	p1 low	
27			pipe	leak (major)		olefin	liquid flashes	
28	More temperature	High intermediate storage temperature	source	t1 high	Higher pressure in transfer line and settling tank	olefin	pressure high; blockage; polymerization	Check whether there is adequate warning of high temperature at intermediate storage
29			pipe	external heat source; hot weather; fire		settler	t1 high; t5 high	
30			pump	pump fluid overheats				
31	Less temperature	Winter conditions	source	t1 low	Water sump and drain line freeze up	pipe	leak (major); brittle frac.	Lag water sump down to drain valve, and steam trace drain valve and drain line
32			pipe	external cold source; cold weather; lagging loss		settler	t1 low; t5 low; drain/sump freeze up	
33	Escape		pipe	leak		olefin	flammable release	
34			pump					
35			nrv.					
36			meter					
37			pipeline					
38			control valve					
39			valve					
40			settler					
41	Maintenance	Equipment failure, flange leak, etc.			Line cannot be completely drained or purged			Install low point drain and N2 purge point downstream of LCV; N2 vent on settler

Additional safeguards were detected, which were not found in the original HAZOP. For instance, due to the substance properties, a "FlareSystem" and a "FlameArrestor" were identified (cf. Table 28, Pos. 40-42, 61-62). Due to risk class **B**, the system automatically identified the safeguard "FailsafeFeedStop" because a high-level alarm is insufficient (cf. Table 28, Pos. 61-62). Furthermore, more pump-specific safeguards have been identified (cf. Table 27, Pos. 1-22). In addition, there are general measures such as "ProvideGroundingOfPlant" (cf. Table 27, Pos. 27-28). In individual cases, slightly different measures are identified with the same purpose. For example, an 'N2 vent on settler' is suggested in the original HAZOP, while the developed knowledge-based system also suggests an "InstallPressureLimitationValve" related to nitrogen blanketing (cf. Table 28, Pos. 63).

Table 31: Comparison of the identified safeguards

Original HAZOP		**Knowledge-based System**	
Description	**Reference**	**Identified classes**	**Reference**
"install low level alarm on settler"	Table 29, Pos. 1	"ImplementLowLevelWarning"	Table 28, Pos. 46-55
"Install kick-back on pumps"	Table 29, Pos. 2	"OverflowValveAndKickbackLine"	Table 27, Pos. 3
"institute regular patrolling and inspection of transfer line"	Table 29 Pos. 4 & 11-16	"RegularPlantInspection"	Table 28, Pos. 35-39
"Install high level alarm on settler and, check sizing relief opposite liquid overfilling, institute locking off procedure for bypass"	Table 29, Pos. 8-9	"PressureReliefValve", "RegularPlantInspection", "FlareSystem", "ValveLockedClose"	Table 28, Pos. 33 Table 28, Pos. 35-38
"Arrange for frequent draining off of water from settler"	Table 29, Pos. 17	"ImplementFrequentDrainingOff"	Table 28, Pos. 31
"Check line FQ and flange ratings, and reduce stroking speed of LCV if necessary"	Table 30, Pos. 20-21	"CheckPressureClassPiping" "IncreaseClosingTimeOfValve"	Table 27, Pos. 29-30 Table 27, Pos. 24
"Install thermal expansion relief on valve section"	Table 30, Pos. 23-25	"InstallThermalExpansionReliefValve"	Table 27, Pos. 23
"Check whether there is adequate warning of high temperature at intermediate storage"	Table 30, Pos. 28-30	"InstallHighTemperatureAlarm"	Table 28, Pos. 41-42
"Lag water sump down to drain valve, and steam trace drain valve and drain line"	Table 30, Pos. 31-32	"InstallHeatTracingSystem	Table 28, Pos. 32
"N_2 vent on settler"	Table 30, Pos. 41	"InstallPressureLimitationValve"	Table 28, Pos. 63

6.3 Summary of the case studies

In this chapter, the developed knowledge-based system was applied within two case studies to evaluate the results. The first case study considers different operation modes within a single-line system involving a hexane tank. The second case study focuses on the propagating scenarios within a *junction system* concerning an olefin feed section. First, the modeling of the plant section in question is described, which includes the equipment entities, substances, site information, and boundary conditions in an input data object. This input data object serves as the basis for the knowledge-based system to perform analyses automatically. There is always a gap between the model and the real world. Accordingly, the level of detail of the models and the consideration of these details within the class definitions within the ontology directly influence the conclusions drawn within the inference algorithm.

Regarding the hexane tank case study, creating the input data object, including reviewing the safety data sheet, took approximately 25 minutes. In comparison, the runtime of the developed knowledge-based system was 81.4 minutes on machine A (cf. Section 5.13.2). Creating the input data object of the olefin feed section case study took approximately 35 minutes, while the runtime was 131.9 minutes on machine A. The time required for creating the input data object depends on the available information and whether the equipment entities of the plant section are included in the equipment library. If the substance information must be modeled from scratch or can be reused, and how complex the signed directed graph is that needs to be modeled.

The time required for a conventional HAZOP study is estimated as follows after consultation with HAZOP practitioners for hexane tank and olefin feed section:

- Preparation: 1 hour,
- HAZOP study: 2.5 hours,
- Follow-up: 1 hour,
- Recommendation processing: 1 hour.

The time required for the HAZOP study and follow-up can be reduced and saved entirely for routine typicals, sing a knowledge-based system. The propagation mode is essential for analyzing the integral behavior of the plant section and the equipment among each other. In the first case study, 12 of 55 scenarios were identified, and in the second case study, 19 of 63 scenarios were identified within the propagation mode.

Since the way the HAZOP analysis is performed directly impacts the number of identified scenarios, a direct comparison of the number of identified scenarios or scenario components does not allow conclusions on the quality of the generated results. Thus, a direct quantitative comparison of the number of identified scenarios does not allow any assessment of the quality of the results. The identified scenarios were interpreted and compared with the reference results obtained from the literature to evaluate the quality of the automatically generated HAZOP worksheets. All significant safety-relevant scenarios must be identified.

The results obtained with the knowledge-based system are equivalent to the reference worksheets for the systems studied, and in specific cases, go beyond the reference worksheets regarding the completeness of safety-relevant scenarios. In contrast to the reference HAZOP worksheets, the worksheets automatically generated by the knowledge-based system contain a risk ranking for each scenario. This is common practice within industry-typical HAZOPs and is required to define appropriate safeguards.

7. Discussion

Computer systems offer the advantage of speeding up the HAZOP process, standardizing results, removing the dependency on the level of knowledge, reducing personnel workload, and allowing more studies to increase safety standards further. In the following, the most significant findings of this work, method-specific implications, relationships to previous research, socio-technical aspects, and limitations of this research are addressed and discussed.

7.1 Recap and implications of the literature review

Approaches to automating aspects of the HAZOP study go back to the 1990s. Initial research attempts used rule-based approaches in the form of expert system shells or other inference engines to identify HAZOP-relevant scenarios or their components, such as causes or consequences (cf. Suokas et al., 1990; Chae and Yoon, 1994; Heino et al., 1995; Khan and Abbasi, 1997a). The major disadvantage of rule-based systems is that the rules must be matched precisely to the problem statement, leading to nested rules and inefficiencies in larger domains (cf. Table 4). Graph-based approaches such as SDGs were used to model equipment and connections (cf. Göring and Schecker, 1993; Vaidhyanathan et al., 1996; McCoy et al., 1999; Wang et al., 2009). Other research groups combined graph-based modeling, such as SDG and Petri nets, to analyze interrelationships on different layers (cf. Zhao et al., 2005b). Other approaches are based on functional modeling where process information, energy, and material flows can be embedded in graph-based modeling (cf. Rossing et al., 2010, Rodríguez and De la Mata, 2012). The advantage is that the graph representation contains both structural and functional aspects. In other prototypes, relationships between process deviations are represented using layered digraphs, while associated causes and consequences are identified using rules (cf. Cui et al., 2008). However, there is no semantic link between causes, consequences, and deviations, and concepts cannot be easily abstracted and transferred. Hence, causes and consequences must be explicitly modeled for each new model. Overall, transferability, abstraction, reusability play an essential role in avoiding isolated solutions. Heterogeneous data structures such as frames, predicate logic, production rules for knowledge representation, Petri nets, and digraphs were combined (cf. Zhao et al., 2005b). Accordingly, such approaches can only be comprehended, extended, and maintained with considerable effort and resources. The knowledge representation or modeling of the processes and plants should not be based on too many different formalisms and paradigms. More recent approaches make use of more advanced reasoning paradigms, such as case-based reasoning, and use contemporary knowledge representation formalisms, such as ontologies (cf. Daramola et al., 2013; Zhao et al., 2009).

Regardless of numerous approaches in the past, no approach or prototype could achieve qualitatively equivalent results compared to well-trained human HAZOP practitioners so far (cf. Chapter 2; Section 4.3). This circumstance can be explained by the fact that more advanced technologies in computer science are available nowadays. Besides, only parts of the HAZOP method were considered within past approaches, e.g., modeling plants or only hazard identification without considering details of the plant. Essential aspects such as the risk assessment of scenarios and selecting suitable safeguards were considered only rudimentarily. In addition, the causation model underlying the scenarios was not always sufficiently considered enabling a computer system to draw appropriate conclusions. As a result, the causal chains of identified scenarios were not always understandable without interpretation. Neglecting these aspects limits the quality of the generated results and prevents their applicability in an industrial environment.

Previous approaches often lack an in-depth comparison of the quality of identified scenarios compared to worksheets prepared by HAZOP practitioners. An exception is AutoHAZID. Some researchers compare the number of identified causes and consequences. However, because HAZOP studies are conducted in different ways, comparing the overall number of scenarios or scenario

components (e.g., causes, consequences) to conclude the quality of the automatically generated results does not appear to be useful. For instance, if a HAZOP worksheet contains all safety-relevant scenarios, it is not inferior to more exhaustive worksheets containing non-relevant scenarios. Finally, the presentation of the results is often far from the typical presentation of conventional HAZOP studies.

7.2 Method-specific implications and significant findings

The HAZOP method is the core methodology for the control of major accidents involving hazardous substances or processes (Seveso-III directive). Safety regulations by the authorities specify procedures, such as overpressure protection (Pressure Equipment Directive 2014/68/EU), explosion protection (ATEX), functional safety (IEC 61508, 61511), fire and water protection. Often HAZOP scenarios serve as the basis for further studies to comply with the safety directives. Due to specific requirements and methodologies, the information gained from HAZOP studies may be incomplete, and additional scenario-specific information must be gathered. Compliance in the chemical industry is an ongoing effort with considerable cost. In the long run, the approach developed in this research may be extended by methodologies required by law that are interfacing with the HAZOP method. For now, the focus of the proposed methodology is on the HAZOP method. The developed and investigated knowledge-based system for digitizing the HAZOP method can be classified with a technology readiness level (TRL) 4 (cf. Tzinis, 2012). Within the scope of this research, basic principles have been observed (TRL 1), technology concepts have been formulated (TRL 2), an experimental proof of concept has been performed (TRL 3), and validation within a laboratory environment has taken place in the context of the case studies (TRL 4). Technological aspects of the developed knowledge-based system (knowledge representation, reasoning, inference algorithm) can be transferred to alternative HAZID methods such as systems theoretic process analysis (STPA) and BLHAZID (cf. Leveson and Thomas, 2018; Seligmann, 2011). In addition, the represented domain can be extended to include aspects from the security domain.

7.2.1 Decomposition of scenarios into their components

A scenario model always reduces reality because only then can causal chains be established (cf. Figure 28). In doing so, both the theoretical model idea behind the HAZOP study and existing industry-typical worksheets were considered. Due to the almost unlimited number of potential scenarios within the chemical industry, an approach was developed to generate scenarios dynamically based on scenario components. Compared to a static approach where each scenario has to be modeled explicitly, it is possible to generate entirely new scenarios by adding single scenario components without invalidating others. Due to the combinatorial possibilities, scenarios can be modeled more efficiently by exploiting the semantic relationships between scenario components. However, the complexity of modeling complete scenarios increases because interactions between scenarios must be considered. In static modeling, semantic relationships between scenario components would not be required because scenarios are modeled in a closed way. Hence, the developed ontology does not contain complete scenarios but statements applying for different scenario components. Thus, scenarios are decomposed into components that are reassembled by the inference algorithm.

Usually, within the industrial practice, a monocausal, linear model serves as the basis for identifying and documenting scenarios. In contrast, a systemic model describes a scenario as a complex network of events, considering socio-technical aspects. Conventional HAZOP scenarios consist of the components: deviation, cause, consequence, likelihood, severity, risk, and safeguards. Starting from a process deviation, causes and then consequences are determined. Since the granularity of events representing the event sequence is essential to identify scenarios automatically,

the introduced concepts **UnderlyingCause** and **Effect** improve the representation of causal chains of scenarios (cf. Figure 28).

Within this research, the cause category is divided into underlying causes and causes, and the consequence category is divided into effects and consequences. Thus, the developed approach represents an extension of the classical view. This makes it possible to identify causes at different hierarchical levels. Effects represent an event between deviation and consequence, allowing to describe scenarios semantically more clearly (cf. Figure 28). The finer breakdown of scenarios is not necessarily required in typical HAZOP studies because the practitioners can interpret, classify, and capture relationships within the scenarios based on professional experience. Additionally, the event sequence of scenarios is formulated in the text, making it unnecessary to use different columns for effects and consequences. While HAZOP practitioners create the scenario using their professional knowledge, all scenario components must be explicitly modeled in a knowledge-based system to assemble the scenario using the inference algorithm. In addition, enough distinguishing features between scenarios are needed so that the correct conclusions can be drawn, since scenarios can run in parallel, e.g., if identical causes lead to different effects leading to different consequences, or if different consequence strings consisting of cause and effect lead to the same consequences. The distinction between effect and consequence simplifies assessing the severity of individual scenarios due to the additional distinguishing feature. For instance, in Figure 85, the severity of scenarios III and IV can be clearly distinguished by the effect, even if the consequence is identical. The **UnderlyingCause** concept refers to systemic issues such as human error, failure of a control loop, or external circumstances. In addition, class definitions of the type **Likelihood** regularly refer to underlying causes. Likelihood information is obtained from the literature or company guidelines (cf. Table 14).

Recommendations represent a comparison between existing and still missing safeguards and measures to reduce the risk to an acceptable level (cf. Section 3.2.1). Recommendations strongly depend on company-specific guidelines, the HAZOP conductors' assessments, and the plant manager or responsible project manager. Within the developed method, existing safeguards are not considered in the analysis (which is in line with HAZOP's usual approach), and accordingly, no recommendations can be made regarding missing safeguards. Therefore, the HAZOP team should make recommendations after comparing automatically identified safeguards and existing safeguards. Typically, the recommendations also include information that should be included in instructions or manuals. Another critical aspect considered is identifying subsequent deviations that arise from effects or causes (backward reasoning). The consideration of subsequent deviations makes it possible to represent complex scenarios that result from other partial scenarios (cf. Figure 66).

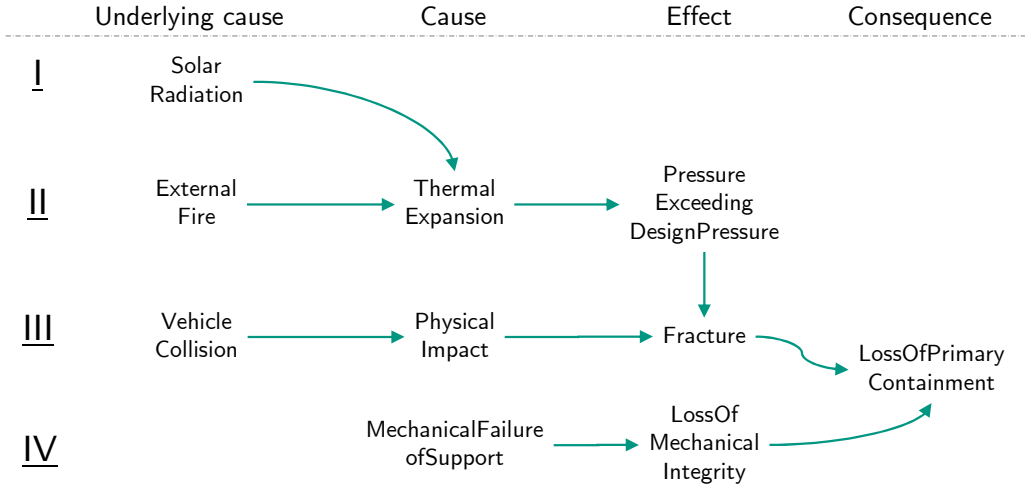


Figure 85: Components of four different scenarios involving a storage tank

Identifying such scenarios is only possible in a backward reasoning step (cf. Figure 65). Operability issues can also be represented based on the scenario components, as shown in the case studies. They are often neglected in typical industry HAZOPs due to time constraints. Besides, the single fault assumption is applied during the scenario identification. Thus, scenarios with two independent initiating events are not considered due to their simultaneous occurrence (very low likelihood). This condition is also called double jeopardy, which HAZOP practitioners do not always consider. Nevertheless, it is common practice in individual companies to consider double jeopardy depending on the hazard posed by the process plant. Only one double jeopardy case is included in the final ontology and may be extended in future work. Additionally, common cause failures (CCF) describe failures due to common causes (cf. DIN EN 61511). CCFs are partially identified but not explicitly related, i.e., no consideration is given to which scenarios can be attributed to the same common cause. For instance, systemic failures, such as "UtilityFailure" or "MaintenanceError", are recognized but not declared as CCF (cf. Table 20, Pos. 4; Table 28, Pos. 61). The developed approach can be extended in future work to identify common causes explicitly.

When analyzing existing HAZOP worksheets, scenarios cannot simply be included in the ontology since they must be decomposed based on the proposed causal model conception of scenarios. Besides, inconsistencies and divergences of knowledge carriers (e.g., experienced HAZOP practitioners) must be compared and cross-checked with other knowledge sources. Accordingly, it is recommended that the knowledge engineer editing the formal ontology understand the domain to be represented.

In addition to the objectives of this work, it is possible to perform causal analyses based on the developed scenario model and the relations between the scenario components based on boilerplates contained in reports. Besides, nonlinear causation models and their representability using the developed knowledge-based system should be further investigated, especially when dealing with complex cyber-physical systems. Examples of methods based on nonlinear causation models are STPA and FRAM (cf. Leveson and Thomas, 2018; Hollnagel, 2012).

7.2.2 The selected level of abstraction of equipment and the environment

Despite the more complex graph-based modeling languages available, SDGs were used to model process plants or process sections. The disadvantage of complex graph-based modeling languages is that the effort for modeling equipment entities increases strongly. The quality of the achievable HAZOP-related results still depends on the knowledge representation and inference method for analyzing the graph. Inference rules have to be modeled based on structural and functional information contained in the graph. The more complex functional relationships are, the more relationships must be decomposed by rules to draw correct conclusions. The process plant must be represented with sufficient detail to identify all safety-relevant scenarios. However, these details must be considered when modeling the scenario components. The more details, the higher the modeling effort for representing scenario components. Increasing the level of detail does not necessarily lead to more complete results but only drastically increases the modeling effort when creating or extending the ontology containing the scenario components. Only if structural and functional conditions are sufficiently covered by inference rules or, in the case of an ontology-based approach, by considering the structural and functional relationships in the class definitions of the ontology, the quality of the results can be increased by increasing the level of detail.

Special attention was paid to investigating and identifying a suitable level of abstraction and detail during the equipment modeling to generalize and transfer concepts. Without abstraction, there may be duplicates in the class definitions within the ontology. Incomplete information on the process plant and overly simplified process equipment models are sources of incomplete and incorrect results. A too detailed description of the involved equipment does not provide an extended benefit but bears the risk of increasing the modeling effort. Considered characteristics of equipment have to be considered when modeling ontology concepts. For example, if details such

as type of connection, such as weld or flange connections, are explicitly represented in the equipment model, they must be considered when modeling scenario components. If only the equipment model is represented in detail, but the level of detail is not included in the ontology classes, there is no benefit. In addition, the effort to create the ontology concepts grows non-linearly since causes, effects, consequences, safeguards, and possibly the concepts for the risk assessment have to be extended by further details from the equipment level.

The overall intention of equipment entities is specified via the intended function, substance task, and mode of operation. The intention of the HAZOP node can be found in the *function description matrix* (cf. Table 19; Table 25). In addition, the function description matrix can be extended by further operation modes, e.g., maintenance, if required. Considering the mode of operation and intended function increases complexity in modeling the ontology classes since the operation mode must be considered in the scenario components. Hence, specific components of the scenario can be further restricted to distinguish them from each other, make more specific and unambiguous statements, and consider functional aspects of scenarios. Especially for ontologies based on the open-world assumption, it is crucial to restrict specific class definitions to distinguish them from general statements. Currently, the intended function refers to the equipment entities. In future work, different states of *subunits* (cf. Section 5.5.4) could still be considered by functional descriptions and operation modes. Since scenarios are not modeled entirely but are composed by the inference algorithm, the operating mode must be consistent throughout the scenario components. If causes propagate from one operating mode to another, this relationship must be recognized by the inference algorithm as part of the propagation of events, which significantly increases complexity.

In addition, the plant's surrounding environment can provide information on the causes (object property: causeInvolvesAmbientInformation) of scenarios. These include environmental hazards that may occur at the site and, on the other hand, traffic that may be present around a plant. Information can be taken from site reports, which address the site conditions. Besides, it is essential to consider whether people are present in the near field of the plant for qualifying the severity of a consequence, which can be set via concepts of the type **BoundaryCondition**. Boundary conditions are used to restrict specific scenario components further to distinguish them from general statements (cf. Section 5.5.8). By default, all potentially possible boundary conditions are active but can be explicitly set if individual available ones can be reasonably excluded. They are set equipment-specific in the input object.

The input object representing equipment entities and connections is currently created manually. Future work could aim to extract equipment and the connections to other equipment automatically. Still, the involved substances, site information, and boundary conditions must be added manually and attached to the project database. Since the required level of detail of the equipment entities goes beyond the information contained in a classic P&ID, details such as fixtures, modes of operation have to be added manually. According to the current knowledge, this step can only be partially automated.

7.2.3 Ontology class and property definitions

The presented results show that ontologies for knowledge representation can help digitize the HAZOP method (cf. Section 4.3). In the course of this work, the following phases of the ontology life cycle were run through: engineering, evaluation, population, evolution, and usage (cf. Section 3.8). The phase usage refers only to the execution of case studies and not industrial implementation. In the initial development phase, the first concepts and relationships between concepts were derived from the competency questions (Section 5.2.4).

Domain-specific findings

Competency questions are not static but change during the life cycle of an ontology. Due to the consideration of implicit relationships, it can also happen that more questions can be answered with the help of the ontology than initially intended. Hence, competency questions only serve as a starting point for ontology engineering. In addition, by analyzing the ontology using Protégé, more advanced competency questions can be answered that are not even provided for in the initial competency questions. For example, relationships can be derived from the hierarchical structures of concepts. In this research, a curated approach is followed, which means the concepts have been created manually, based on expert knowledge aggregated from different sources, such as technical literature, expert interviews, and existing HAZOP worksheets. The class definitions within the ontology were continuously improved and refined during the evaluation of the generated HAZOP worksheets. The extension of the ontology to include method-specific aspects such as risk, modes of operation, and the refinement of the level of detail in the representation of equipment resulted in continuous changes to the class definitions.

Due to the versatility of chemical process engineering systems, the number of possible scenarios is practically unlimited, leading to a *content completeness problem* (cf. Section 5.1). Therefore, scenarios are not modeled in their entirety within the ontology but are composed of a combination of scenario components (underlying causes, causes, effects, consequences) by the inference algorithm depending on the input data. Since the ontological model relates scenario components, additional scenarios can be created by only adding individual components, such as causes or effects, resulting in entirely new scenarios. Dynamic scenario generation can mitigate the *content completeness problem*.

The **BoundaryCondition**s concept was introduced to distinguish specific scenario components from each other unambiguously. During the development of the proposed ontology, it turned out that with an increasing number of modeled and related classes, implicit relationships between concepts can lead to unpredictable inferences correctly drawn by the reasoner but incorrect or not foreseen within the modeled domain. The extent and the case-specific completeness of the ontology were iteratively checked and adjusted. The accuracy, completeness, relevance, and reliability of the generated HAZOP worksheets serve as a decisive indicator for the performance of the developed knowledge-based system consisting of the ontology and the inference algorithm (cf. Section 0). Completeness is achieved if all safety-relevant scenarios are recognized.

OWL 2 language-specific findings

When modeling concepts using the OWL 2 DL ontology language, there is always the question of whether to model concepts with classes or individuals (instances of classes) (cf. Figure 15). The type of modeling depends on the knowledge engineer, the domain to be represented, and the application in which the ontology will be used. In case of ambiguity Stevens and Sattler (2013) recommend using classes instead of instances. In this work, all concepts are modeled as classes while instances (individuals) of the scenario components, equipment entities, substances and substance properties, and site and boundary conditions are dynamically created when executing the knowledge-based system.

Specific scenario components have been modeled as classes to formulate additional axioms to constrain their use. Defining axioms is essential because they ensure the plausibility of the overall scenarios that a reasoner assembles from modeled scenario components. The ontology language OWL 2 DL used within this work proved to be sufficiently expressive to represent relevant concepts. Within the class definitions, no general rules in IF-THEN statements were required, and accordingly, no SWRL rules were needed (cf. Section 3.8.1). When using property restrictions (cf. property axioms, Figure 15), such as *transitive* or *asymmetric restriction*, care must be taken to ensure compatibility with each other. For example, transitive properties cannot be disjoint within OWL 2 DL (cf. Schneider *et al.*, 2013).

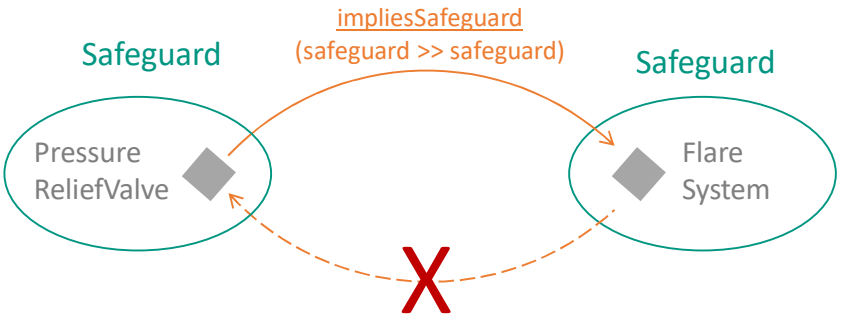


Figure 86: Example of the property axiom: AsymmetricProperty

No property restrictions were used for object properties with a different domain and range within this work. For object properties with identical domain and range, such as impliesSafeguard (cf. Figure 55) or isSubsequentConsequence (cf. Figure 53), the property axiom *AsymmetricProperty* was used to prevent false inferences. Figure 86 shows the effect of the property axiom *AsymmetricProperty*. This axiom limits that one Safeguard implies another Safeguard but not vice versa. The property axiom *AsymmetricProperty* is not available in OWL 1, and OWL 2 is required. Data properties, such as *hasMaximumOperatingPressureInBarGauge,* are provided with the property axiom *FunctionalProperty* (cf. Appendix Listing I-4). Hence, their range may contain at most one element.

Duplicates must be avoided in the class definition to avoid unpredictable inferences, except empty forward declarations. Besides the reduced maintainability, local ontology class definitions or restrictions in one component could spread and cause unwanted global effects. Also, cyclic classes should be avoided within the class hierarchy (cf. Noy and McGuinness, n.d.). For example, this is the case if class A has a subclass B, which is a superclass of A. Cyclic class definitions can lead to performance losses during reasoning (cf. Liebig, 2006). Besides, it could be observed within this work that the use of inverse properties within the developed ontology structure increases the reasoning time of the HermiT reasoner (cf. Section 5.13.2). Concerning the industrial utilization and scalability of knowledge-based systems, graph databases for knowledge representation, such as Neo4j as an alternative to quad stores (OWL 2), is recommended to be investigated in more detail (cf. Neo4j, n.d.).

Application-specific findings

The ontology development was done programmatically to use best practices from software engineering like version control, refactoring, and unit tests. In addition, it is possible to use professional integrated development environments (IDEs), e.g., PyCharm, in the development context. The maintenance and extension of the ontology require profound knowledge in knowledge engineering an understanding of the architecture of the ontology. Besides, domain knowledge and understanding of how scenario components are modeled and scenarios are assembled by the inference algorithm are required to extend the ontology. Inadequacies within the class definition are not already directly apparent during the class definition but only become apparent after the inference process in the context of the inference algorithm. The advantage of programmatically creating the ontology is that unit tests can be created for the class definitions after creating class definitions. Nevertheless, unit tests cannot entirely eliminate this issue since they are intended for testing individual class definitions. Due to the combinatorial possibilities of the scenario components (underlying cause, cause, deviation, effect, consequence), unit tests cannot be performed for every theoretically possible scenario. Nevertheless, the usefulness of unit tests is undisputed and has proven to be particularly important in building the application-oriented class definitions of the ontology to understand the impact of local changes on other class definitions. The overall ontology was regularly checked for consistency errors, e.g., incorrect class definitions, using the

Protégé editor during development. After the further extension of the ontology, black-box tests could be used to analyze plant parts without knowledge of the inference algorithm and to evaluate the quality of the generated HAZOP worksheets. Future research can extend the ontology using a semi-automatic approach (cf. Single *et al.*, 2020d).

Without knowing the overall context of the architecture, domain-specific implications may be incorrectly drawn by the software component because class definitions are equated to other class definitions according to set theory (cf. Section 7.2.4). Accordingly, further research is needed to determine whether it is possible to use forms and a mechanism to incorporate concepts into the overall ontology structure without drawing incorrect conclusions. A mechanism would be required to enable a user (HAZOP practitioner or domain expert) to add or edit knowledge concepts. Currently, it does not seem feasible that users can directly intervene in the knowledge concepts of the system. In addition, it has to be investigated whether ontology learning methods can automatically insert class definitions due to the complex and 23 object properties between scenario components, risk category, and safeguards. During an industrial implementation, the usage of ontologies must be governed within an ontology management system. Among other things, the following aspects should be considered:

- storage, filing, versioning of the ontology,
- dealing with data sources and inconsistent information,
- maintenance, extension, and adaptation of the ontological model,
- use of reasoners, query interface, programmatic implementation of reasoners,
- review instances and approval processes, ensuring traceability and auditability.

Section 5.10 describes the proposed approach of automatically generating a knowledge graph based on a schema ontology (cf. Single *et al.*, 2020d). The final ontology proposed within this dissertation consists of 625 classes and 85 relationships (object and data properties). Accordingly, there is a significantly higher complexity due to numerous relationships between concepts. Since scenarios are derived from scenario components within the inference process, it seems questionable if individual scenario components can be created based on ontology learning and assembled into plausible and complete scenarios during inference. In this work, interaction effects between individual scenario components and the overall scenario could partly only be identified by systematically analyzing the automatically generated HAZOP worksheets. In the future, ontology learning methods may contribute that ontologies and knowledge-graphs can evolve autonomously based on new information.

7.2.4 Overlapping class definitions during risk assessment

As shown in the worksheets of the case studies, the risk assessment of the scenarios may result in two risk classes being identified for individual scenarios (cf. Table 21, Pos. 43). The risk class is determined based on the likelihood and severity of the scenario, which depends on the scenario components, equipment, and hazard classes of the substances. Since different substance properties and equipment can lead to different severity categories, multiple severity categories may be identified based on multiple substance properties held by a substance. This conclusion is semantically correct because **Risk** class definitions overlap. This relationship will be explained through an example involving a storage tank and an LOPC with the following scenario components (Table 21, Pos. 43):

- Underlying cause: “VehicleCollision”,
- Cause: “PhysicalImpact”,
- Effect: “Fracture”,
- Consequence: “LossOfPrimaryContainment” (LOPC).

It is assumed that there are people in the near field of the plant (**PersonnelPresentInTheNearField**). Since this scenario involves the substance hexane, the hazard classes are (cf. Table 18):

- **FlammableLiquidCategory2**,
- **SkinCorrosionIrritationCategory2**,
- **ReproductiveToxicityCategory2**,
- **SpecificTargetOrganToxicitySingleExposureCategory3**,
- **SpecificTargetOrganToxicityRepeatedExposureCategory2**,
- **AspirationHazardCategory1**,
- **HazardousToAquaticEnvironmentCategory2**.

The following are the essential excerpts from the formal class definitions using DL syntax. The cause "PhysicalImpact" leads to the likelihood category **Possible** (cf. Appendix Listing O-1):

$$Possible \equiv Likelihood \sqcap \exists likelihoodInvolvesCause.VehicleCollision \tag{14}$$

Due to the different hazard classes of hexane, the severity categories **Severe** and **Serious** result for the example scenario (cf. Appendix Listing O-3):

$$\begin{aligned} Severe \equiv\ & Severity \sqcap \exists isSeverityOfConsequence.LossOfPrimaryContainment \sqcap \\ & \exists underlyingcauseRequiresBoundaryCondition.PersonellPresentIntheNearField \sqcap \\ & \exists severityInvolvesSubstance.(\exists hasHazardClass.(FlammableLiquidCategory1 \sqcup \\ & \qquad FlammableLiquidCategory2)) \end{aligned} \tag{15}$$

$$\begin{aligned} Serious \equiv\ & Severity \sqcap \\ & \exists isSeverityOfConsequence.LossOfPrimaryContainment \sqcap \\ & \exists severityInvolvesSubstance.(\exists hasHazardClass. \\ & \qquad (FlammableLiquidCategory3 \sqcup \\ & \qquad ReproductiveToxicityCategory2 \sqcup \\ & \qquad SkinCorrosionIrritationCategory2 \sqcup \\ & \qquad SkinCorrosionIrritationCategory3 \sqcup \\ & \qquad SpecificTargetOrganToxicitySingleExposureCategory3 \sqcup \\ & \qquad SpecificTargetOrganToxicityRepeatedExposureCategory2 \sqcup \\ & \qquad SpecificTargetOrganToxicitySingleExposureCategory3 \sqcup \\ & \qquad SecificTargetOrganToxicityRepeatedExposureCategory2)) \end{aligned} \tag{16}$$

The likelihood category **Possible** and the severity categories **Severe** and **Serious** yield in the risk categories **B** and **C**, based on the risk matrix (cf. Figure 6):

$$B \equiv RiskCategory \sqcap \exists involvesSeverity.Severe \sqcap \exists involvesLikelihood.Possible \tag{17}$$

$$C \equiv RiskCategory \sqcap \exists involvesSeverity.Serious \sqcap \exists involvesLikelihood.Possible \tag{18}$$

Accordingly, two risk classes result for the scenario described above based on the intersections presented. Due to the combinatorial variants of hazard classes belonging to specific substances, multiple severity categories may be inferred. The same applies to combinatorial variants of cause and underlying cause, determining the likelihood. Based on the risk categories within the risk matrix, two different risk classes may be identified for one scenario. In this case, expert judgment from HAZOP practitioners is required to determine a final risk category for the scenario.

7.2.5 Ontology evaluation using ontology metrics

Different ontologies have been designed and investigated within this research, and the final ontology presented within this dissertation has evolved. In addition to the qualitative and outcome-oriented evaluation of the ontology, ontology metrics were used for quantitative evaluation.

Tartir *et al.* (2005) proposed multiple metrics and developed the ontology evaluation tool OntoQA. The evaluations within this thesis were performed using the web-based tool OntoMetrics (cf. Lantow, 2016). A distinction can be made between the categories base, schema, and knowledgebase metrics. Base metrics provide an overview of the extent of the ontologies. Schema metrics can be used to evaluate the design of the ontology and the inherent capability for knowledge representation. Knowledgebase metrics are a measure for the utilization of schema classes. The metrics have limited meaning in absolute terms but can be used to compare different ontologies. The ontology metrics of three ontologies developed and investigated in this research work are shown in Table 32. The chemical accident ontology was developed during the research described in Section 5.10; and includes only 7 classes, 28 object properties, 8 data properties, and all accident scenario components, and all accident database entries were modeled as individuals (class instances). The schematic HAZOP ontology was developed to demonstrate basic concepts and contains 163 classes, 33 object properties, 1 data property, and 47 individuals. The base computer-aided HAZOP ontology (CAHO), described in detail in this dissertation and used by the inference algorithm to generate the results within the case studies, contains 625 classes, 70 object properties, 15 data properties, and no individuals. The fact that this ontology does not contain individuals can be explained because all concepts have been modeled in classes. These classes are automatically generated and processed by inference algorithms. The first two ontologies are not intended to be evaluated automatically. The number of instances created depends directly on the system analyzed and the input and is not static.

Table 32 shows that schema-related metrics remain constant within the CAHO as the instance-related metrics change. The equations for calculating the ontology metrics are shown in Appendix T. The relationship richness (RR) is a metric to quantify the diversity of relationships. Tartir *et al.* (2010) state that an ontology mainly composed of inheritance relationships contains less information compared to mixed relationships. Accordingly, it has mainly the features of a taxonomy. An ontology with only inheritance relationships (is_a) would result in an RR of 0. In Table 32, the chemical accident ontology (CAO) has the highest RR value of 0.51, which is understandable given the small number of classes (C: 7) and the number of relationships (object properties: 28, data properties: 8). The schematic HAZOP ontology (SHO) has a significantly lower RR value of 0.11 since it was created for demonstration purposes only. The CAHO-B ontology has a RR value of 0.368, which can be explained by the fact that besides numerous relationships between core classes, many inheritance relationships (is_a) were used when creating the classes to describe substance properties and to represent all groups and subgroups of plant parts.

The inheritance richness (IR) metric indicates the distribution of information across hierarchical levels (parent and child classes). A low IR score indicates an ontology covering a specific topic in detail (cf. Tartir et al., 2010). On the other hand, a high IR score indicates that the ontology represents a broad topic with lesser detail (cf. Tartir et al., 2010). Tartir et al. (2010) note that the IR is a good indicator of how well knowledge is grouped into categories and subcategories. The CAO has an IR value of 3.86, the highest of all proposed ontologies because the seven classes are on the same hierarchical level (cf. Table 32). The CAHO has the lowest IR value of 0.99, suggesting that the ontology is a deep ontology where a specific domain is covered in detail. This observation is also consistent with the goal of the ontology to provide knowledge for the automatic generation of HAZOP worksheets. The average population (AP) metric is calculated from the number of instances and classes and describes the average distribution of instances (cf. Appendix T). The AP metric should be used in conjunction with class richness (CR) and corresponds to the ratio of classes with instances and the total number of classes (cf. Appendix T). The specific scenarios within the CAO were modeled as individuals can be directly seen in the AP value of 202.71. There are 202.71 individuals per class on average, resulting in a CR value of 1.

Table 32: Comparison of ontology metrics

Description	Chemical accident ontology	Schematic HAZOP ontology	Base CAH ontology	CAH ontology - case study I	CAH ontology - case study II
Acronym	CAO	SHO	CAHO-B	CAHO-1	CAHO-2
Ontology described in	Single et al. (2020b)	Single *et al.* (2020b)		This dissertation	
Base metrics					
Class count	7	163	625	625	625
Individuals	1419	47	0	1800	2581
Object properties	28	33	70	70	70
Data properties	8	1	15	15	15
Schema metrics					
Relationship richness (RR)	0.51	0.11	0.367	0.367	0.367
Inheritance richness (IR)	3.86	2.66	0.998	0.998	0.998
Knowledgebase metrics					
Average population (AP)	202.71	0.288	0.0	2.88	4.129
Class richness (CR)	1	0.074	0.0	0.274	0.292

Hence, 100 % of the classes used have at least one individual. Due to its conceptual nature, the SHO contains only exemplary instances, resulting in a low AP value of 0.288 and a CR value of 0.074. Instance-related metrics reflect the circumstance that the instances within the CAHO ontology are dynamically generated by the inference algorithm depending on the analyzed system. In the first case study, more instances per class are generated on average (case study 1: AP: 2.88; case study 2: AP: 4.129). Knowledge utilization is slightly higher within the second case study (CR: 0.292) than the first (CR: 0.274). The results show that ontology metrics can be used to quantitatively compare the structures of different ontologies from different points of view, such as schema or knowledgebase. Thus, it is possible to systematically track changes in the overall structure within the ontology development process. Since the inference algorithm automatically generates individuals within the developed approach, knowledgebase metrics provide information about the extent to which ontology concepts are used. Thus, they describe the utilization of classes defined in the schema, expressed in particular by the class richness (CR). After each run, automatically created instances are deleted again.

7.2.6 Importance of topology-specific propagation and subsequent deviation

The topology of the plant or plant section determines whether the plant is divided into different sections, which are analyzed separately under consideration of the interfaces. Besides, the topology determines the order in which plant sections are examined. Even though the complete set of deviations is applied in the equipment-based analysis, only the propagation mode provides plausible causes for specific scenarios (cf. Section 5.14). Only the propagation mode recognizes material and energy flow between equipment entities. Propagation-based strategies are crucial in non-linear plant section topologies, such as recycle streams. In the automatically generated HAZOP worksheets, scenarios identified within the propagation mode are marked with a 'True' in the propagated column (cf. Table 20). In the first case study, 12 out of 55 scenarios were identified within the propagation mode. 19 out of 63 scenarios have been identified within the propagation mode within the second case study. Thus, the proportion of scenarios identified within the propagation mode is between 21.8 and 30.2 %. The share of scenarios identified within the

propagation mode tends to increase with the complexity of the plant. Another aspect that has turned out to be important is that scenario components, such as effects, can lead to subsequent deviations. By rigorously applying deviations to equipment, one does not necessarily arrive at these scenarios. Due to the causal relationship in the semantic model that effects are identified based on deviations, deviations cannot be derived directly from effects. Subsequent deviations are identified based on effects, using backward reasoning. Ergo, an effect explains the deviation, which is not possible within the semantic model in the ontology. Due to the disjointness axiom between cause and effect, an effect cannot also be a cause. Hence, the backwards reasoning step is necessary based on case-based reasoning. For example, the effect "Cavitation" results in the deviation **High-Vibration**.

7.3 Relationships between findings and previous approaches

Due to the novel approach developed and investigated within this research conclusions, recommendations for further research arise compared to previous research approaches. Besides, there are parallels between approaches in the past and the method developed within this research. For example, other research groups have developed model libraries as part of their prototypes to represent plant sections and equipment and to be able to apply their methods (cf. Vaidhyanathan et al., 1996; McCoy et al., 1999; Zhao et al., 2005a; Cui et al., 2010). However, since these model libraries differ fundamentally in structure, knowledge representation formalism, and level of detail, they cannot simply be adopted or transferred. Furthermore, previous approaches did not sufficiently consider the intended function and operating mode, which play a significant role in conventional HAZOP studies.

7.3.1 Safeguards and risk assessment

Researchers in the past mostly neglected safeguards within their approaches. Without a risk assessment, the scope and extent of safeguards cannot always be adequately determined. So as part of the digitization of the HAZOP method, a risk assessment must also occur. In the KROSA approach, the risk class was considered, but not the dependencies between scenario components, e.g., cause, effect, likelihood, and severity, from which risk classes are derived (cf. Daramola et al., 2013). This view leads to a static scenario-based view, which prevents the risk assessment of scenarios from being abstractable and transferable. In AutoHAZID, only consequence-related severity is considered by estimating the number of injuries and fatalities. However, the hazard ranking information is not included in the HAZOP tables (cf. McCoy, Wakeman, Larkin, Chung, et al., 1999). The PHASuite approach deals only rudimentarily with the severity of scenarios, as specific consequences are only assigned a particular severity category (cf. Zhao et al., 2005a).

Risk assessment is essential in industrial HAZOP practice to focus on particularly critical scenarios. Therefore, semantic relationships between likelihood, severity, risk, and the other components of the scenario and equipment and substance properties were considered in this work to determine a risk class for each scenario automatically. Hence, an industry-typical risk matrix was represented within the ontology. Furthermore, relationships were provided to define proposals for safeguards depending on the risk class. For example, different safeguards mitigate the effect of "Overfilling" depending on the risk class:

- Risk category: C or D → safeguard: "InstallHighLevelAlarm", "CheckOverflowLocation",
- Risk category: A or B → safeguard: "FailsafeFeedStop".

The accuracy of the automatically identified safeguards was demonstrated compared to the safeguards in the reference worksheets (cf. Table 24, Table 31). Identifying relevant safeguards is a significant step forward compared to previous research approaches. In case further information

is needed for individual safeguards, these can be appended to the class definitions in the ontology in the form of annotations (cf. Figure 15). Annotations can be queried with the help of reasoners. The event sequence of a scenario can typically be interrupted by different safeguards. Accordingly, selecting safeguards is always a matter of judgment that must be made by the plant manager or the responsible project manager. The developed KB proposes plausible safeguards for specific scenarios from which a user can choose.

7.3.2 Propagation of events

The propagation of events within AutoHAZID is based on qualitative models which describe the functional relationships between input and output variables (pressure, temperature, flow) of each piece of equipment. AutoHAZID is based on the approaches of Parmar and Lees (1987b). They note that the same deviation is propagated without change in most cases, and other deviations arise only in specific equipment. The unit-specific functional relationships are combined with connectivity information for representing a connected network of relationships. The fault propagation within AutoHAZID is implemented SDGs. The AHA prototype of (Kang *et al.*, 2001) is also based on so-called *unit behavior models*, which describe the relationship between input and output variables of equipment. It is used for modeling fault propagation relationships. The approach is similar to the AutoHAZID approach in this aspect. Functional relationships between input and output deviations are represented using the expert system shell G2.

In the proposed knowledge-based systems, input-output relationships are inferred using CBR. Hence, deviations that arise from specific causes or effects are identified using a specifically implemented case-based reasoning approach. Here, the attributes for case discrimination also include the intended function, apparatus, and the state of aggregation of the involved substance. Using CBR has the advantage that a conclusion can be drawn based on incomplete knowledge since it does not require an exact model (cf. Section 5.13.2). Thus, new equipment entities can be processed based on structural or functional similarities without creating new models. Compared to functional behavior models of AutoHAZID and AHA, this approach brings more flexibility, and the maintenance effort is lower because only new cases must be added instead of modeling and adapting existing functional models. Adding new, more suitable cases improves reasoning since the determined similarity measure is higher for the more suitable case compared to older cases. Within the developed propagation algorithm, partial scenarios that qualify for propagation are appended to an equipment-specific list (propagation stack) evaluated in the propagation mode of the inference algorithm. A partial scenario that has no immediate effects or consequences in the current equipment may lead to effects and consequences in adjacent equipment (cf. Section 5.14). In addition, the plant section topology was considered within the propagation algorithm. Hence, specific topologies are captured to improve the recognition of the structural relationships between equipment and the propagation paths of events. Based on the results of this research work, it can be concluded that the number of scenarios identified in propagation mode increases for more complex topologies. This circumstance needs to be investigated further.

7.3.3 Evaluation of the quality of automatically generated results

The quality of automatically generated HAZOP worksheets is evaluated under the criteria: *accuracy*, *completeness*, and *relevance* (cf. Chapter 2 (9)). The *reliability* criterion is ensured by the deterministic approach of the inference algorithm. Parmar and Lees (1987b) mentioned that computer methods generate a more exhaustive list of causes and consequences than HAZOP practitioners. The point raised by Cui et al. (2008) that automatically generated results are significantly more extensive, and most of them are meaningless compared to the results of experts could not be confirmed within this work. McCoy *et al.* (2000) noted that qualitative models combined with qualitative reasoning could lead to infeasible scenarios. Therefore, the AutoHAZID approach uses a fault path validation and a *process dependency* filter to resolve infeasible scenarios (cf.

McCoy et al., 2000). Within AutoHAZID, the equipment-specific approach means that each valve, e.g., isolation valve, is considered leading to an excessive number of identified causes. Subsequently, these causes are filtered out since they do not provide new aspects and lead to repetitions in the worksheets (cf. Wakeman et al., 1997). When using filters, the aim is to fulfill the quality criteria *relevance* (cf. Chapter 2 (9)).

In the automatically generated worksheets of the proposed approach, each scenario occurs only once, while each scenario differs in at least one scenario component. Also, there are no cross-references to other scenarios, as causal chains (subsequent events) are presented directly (cf. Section 5.13). In this research, the causal model underlying the HAZOP method was examined in detail and extended by the scenario components **UnderlyingCause** and **Effect**. In addition, the disjointness axiom between scenario components prevents 'trivial' causes or consequences, i.e., causes or consequences that are equal to deviations (cf. Section 6.2.5). Hence, the proposed method provides more distinguishing features for describing scenarios whereby the quality criterion *accuracy* is fulfilled (cf. Section 5.2.3, Chapter 2 (9)).

The focus of this work was on the representation of causal relationships and the features of scenarios, the choice of an appropriate abstraction level of process equipment, substances, and the engineering-oriented modeling of specific class definitions within the ontology, redundant or meaningless results in the HAZOP worksheets could be avoided. Hence, the quality criterion *relevance* of the generated worksheets is higher compared to past approaches. The results of literature approaches and prototypes were compared to the forms of presentation of this work. Table 33 shows the form of presentation of the AutoHAZID and AHA results. These are excerpts of the results of the olefin feed section, which was covered in Section 6.2. The AHA results consist of the equipment involved, such as PIPE-4, and acronyms representing causes, such as completely blocked (CB), fracture (FR), failed closed (FC), stuck closed (SC), or pump failure (PUMPF). Consequently, acronyms and abbreviations must be translated, and the representation must be interpreted to derive individual scenarios. Further excerpts from results generated by Rahman *et al.* (2009) other research groups are presented in Table 34. Due to the way causal relationships between scenario components are modeled, specific scenarios have to be interpreted. For instance, in the second scenario, the deviation 'more temperature from V1' has the cause 'pump failure', and the consequence 'reactor rupture' (cf. Table 34). A too high temperature in a reactor does not directly lead to a rupture of the reactor vessel. Hence, an event between cause and consequence is missing. Within this scenario, it becomes apparent that the causal relationships between cause and consequence cannot be understood without a deeper understanding of the representation of the scenario. Accordingly, the quality criterion *accuracy* is not entirely fulfilled because the causal chain of the scenario is not fully represented.

The *accuracy* of the worksheets generated by the knowledge-based system developed within this research goes beyond that since each scenario is explicitly presented, including the substance involved, the intended function, and a complete description of the sequence of events (cf. Table 29; Table 30). Currently, the class names are used directly in the HAZOP worksheets. If extended descriptions and details of the scenario components are desired, they can be annotated and displayed via *rdfs:label* (natural language label). Compared to previous approaches, large intersections of scenario components despite different deviations were avoided by clearly assigning specific deviations to scenarios. For instance, the deviation 'low flow' and 'no flow' have multiple causes in common (cf. Table 29). The original idea behind the deviations is to stimulate creativity in identifying scenarios rather than stoically applying every available deviation. Besides, causal chains of the scenarios were carefully modeled in this work and inferred using forward and backward reasoning to meet the quality criterion *accuracy* (cf. Figure 66). Due to the granularity of the scenario components, the presentation of scenarios is more understandable compared to previous approaches.

Table 33: Comparison of the form of presentation of different approaches

AutoHAZID – (McCoy *et al.*, 2000)		
Deviation	**Causes**	**Consequence**
BufferTank noFlow in	levelControlValve1 closed, valve8 etc. closed; valve2 etc. closed, feedInlet no flow upstream	bufferTank loss of level, gas breakthrough to downstream units; bufferTank loss of level, gas breakthrough to downstream units, pumpJ1a cavitation
AHA – (Kang *et al.*, 2001)		
NO FLOW in PIPE-4	PIPE-4: CB, FR CV-1: FC, SC CECKV-1: FC, SC PUMP-1: PUMPF	SETTLER-1 Running Dry

Table 34: Excerpt of HAZOP worksheets from the ExpHAZOP$^+$ approach

ExpHAZOP$^+$ – (Rahman *et al.*, 2009)				
Item	**Connect. equip.**	**Deviation**	**Causes**	**Consequence**
Temperature sensor (TS)	From equipment HE pipe 2	More temperature from V1	Temperature sensor failure	Pump load increase
Reactor	From equipment TS pipe 3	More temperature from V1	Pump failure	Reactor rupture

7.4 Socio-technical aspects and implications for the future

Integrating a knowledge-based system into the human-centered HAZOP process creates a socio-technical system consisting of a technical subcomponent and a social component. According to Manzey (2012), automated systems are socio-technical systems whose reliability depends on the interaction between humans and automation. Bainbridge (1983) described that automation approaches are intended to relieve, support, and replace experts. On the other hand, the responsibility for abnormal conditions is left to the operator/user of automated systems as the "ironies of automation". Sheridan (1992) used the term *human supervisory control* to describe the task that the human expert initiates, monitors, and intervenes in automated systems' results, recommendations, or actions. Since the HAZOP method is a creativity method, the question arises as to whether it makes sense to digitize or automate it. Accordingly, in the context of the investigation and development of a knowledge-based system, the influence on the working methods of HAZOP practitioners, the integration into existing workflows and processes, and the handling of automatically generated results must be considered.

7.4.1 Trust in automation

Manzey (2012) states that the main factors influencing trust in automated technical systems include system reliability, traceability, and perceived usefulness. Sheridan formulates various attributes of trust: *reliability*, *robustness*, *familiarity*, *understandability*, *explication of intention*, *usefulness,* and *dependence* (cf. Sheridan, 1992, pp. 347-348). These attributes should be considered to build trust in automated systems. The *reliability* attribute can be satisfied by automatically obtaining reliable results for defined equipment parts and processes in a deterministic manner that HAZOP practitioners accept. *Robustness* can be achieved by ensuring the validity of the input object and by providing fallback routines within the software in case operations fail. In addition,

this includes a specific range of variations in the input object for which plausible results are still obtained. *Familiarity* can be achieved by using industry-standard terms in the worksheets. Potential users should understand the technical aspects of knowledge-based systems and their fundamental aspects. By enabling HAZOP practitioners to extend such systems in the context of HAZOPs by their technical knowledge, e.g., by using forms by which software agents automatically extend the application-oriented part of the ontology, understanding can be achieved. An *explication of intent* can be achieved by implementing a software deployment through an organizational process and systematically monitoring it during the implementation phase, where expectations and objectives are clear to all parties involved. The *usefulness* is undoubtedly given by the time savings, simplified analysis, reduced routine activities, and transparency in case of revisions of plant sections. In one aspect, computer systems directly improve the quality of HAZOP Studies by taking time pressure out of the study.

Additionally, the personal attitude towards technology, how people deal with new technologies, and their cultural approach play a decisive role in how computer systems are trusted and differ in regions. Manzey (2012) distinguishes between *excessive trust* and *lack of trust*. While excessive trust (or *automation complacency*) can lead to insufficient monitoring and loss of situational awareness and overlooking errors, a lack of trust can lead to low usage, no relief to experts, and ignored notices (cf. Manzey, 2012). Mosier and Skitka (1996) use the term automation bias in the context of excessive trust where operators fail to notice problems because the automated aid fails or when an automated decision aid directive is followed inappropriately. In terms of HAZOP studies and knowledge-based systems, this could mean that the scenarios within the automatically generated HAZOP worksheets are not critically questioned but accepted even though they are not appropriate, or that scenarios are not considered further even though they are valid.

7.4.2 Situational awareness and loss of skill

In addition, the way experts work must be considered. Manzey (2012) described three problems related to the usage of automated systems:

(1) Excessive trust in automation,

(2) Maintaining appropriate *situational awareness*, which means that a person is aware of what and why something is happening and how to assess the situation,

(3) Loss of skills by repeatedly using intelligent systems.

These general observations and thoughts can be transferred directly to HAZOP studies. Situational awareness includes assessing possible applications of software, e.g., which process plants or nodes can be analyzed by computer systems and which have to be analyzed by human experts. Furthermore, the operator must recognize the wrong conclusions of computer systems and instead trust and accept their conclusions or findings as HAZOP results. HAZOP practitioners must not lose their problem-solving competence and ability to conduct studies to maintain situational awareness. A loss of these abilities and knowledge can be described as a *loss of skills*. A reviewing role of HAZOP practitioners may lead to the loss of cognitive skills because of the convenience of accepting automatically generated results. HAZOP studies must still be regularly performed to prevent the loss of skills despite using computer systems. Sheridan (1992) addresses that skill provides the human expert with "a sense of dignity and self-respect." This circumstance applies especially to HAZOP experts that need years of training and professional experience to do their jobs. Experienced HAZOP conductors, in particular, may reject such systems because the value of their work is devalued if a computer system can perform the task.

7.4.3 Implementing a knowledge-based system into the HAZOP process

Digitization of the HAZOP method alone is not sufficient because computer systems must be integrated into the human-centered HAZOP process. Hence, the role of participants in HAZOP studies must be considered. The study leader has organizational tasks, conducts preparation and follow-up, leads the discussion, documents the results, and distributes further tasks. The other participants provide the required expertise and experience. Through the discussions in the group and the consensus of the validity of scenarios, the group fulfills its duty of care. Knowledge-based systems can be used as part of HAZOP preparation. For example, automated worksheets could be sent to participants in advance to discuss the content at the HAZOP meeting, reducing the time required for the study. As part of the training of new HAZOP practitioners, systems can be used to benchmark results. Another application would be to use knowledge-based systems to analyze typical equipment or typical plant sections which occur repeatedly and are designed in the same way, while the HAZOP team takes care of the interfaces or more complex plant sections.

Using knowledge-based systems, human experts could shift towards the supervisory control of knowledge-based systems because automatically generated results must be evaluated and accepted or rejected. Sheridan (1992) used the term *supervisory control* and described five generic supervisory functions of the human supervisor: (1) planning of the tasks, (2) programming the computer system, (3) monitoring the automatic action and detect failures, (4) intervening, and (5) learning from experience. Sheridan (1992) also addresses that automation is often intended to reduce the workload, but monitoring computer systems initially creates workload. Additionally, from an engineering psychology perspective, it must be considered that supervisory tasks may not be as rewarding as coming up with new ideas, leading to dissatisfaction.

The supervisory control aspect is especially true in the introductory phase of such systems because there is no experience. Hence, the HAZOP team serves as a 'peer group' of the knowledge-based system. After the first test runs and the evaluation of the completeness of automatically generated worksheets, the ontology must be adapted and enhanced to newly created plant sections and processes. It must be ensured that the adjustments do not distort existing definitions and their interpretation by reasoners. To achieve this, unit tests for individual class definitions are essential. In the medium term, the benefits of such systems must compensate for the additional costs incurred in the initial phase. Therefore, the workload reduction achieved by knowledge-based systems must be more significant than the additional monitoring and implementation effort. The resubmission process of HAZOP studies can certainly be simplified by making changes to the input data object, having the knowledge-based system perform the evaluation, and comparing changes made in the HAZOP worksheets directly.

Typically, HAZOP studies are resubmitted every five years, but company-specific clean sheet analyses may be performed every ten years. Depending on the composition of the team, the results may differ from those of previous studies. Even when knowledge-based systems are used, different scenarios may arise from the results of HAZOP teams. Therefore, potentially controversial scenarios are not an exclusive problem of computer systems. When creating a process that defines how to use and interact with a knowledge-based system, it must be determined how controversial results will be managed, e.g., whether such results will lead to the adaptation of ontology concepts. Differences in opinion, or disagreement of automatically identified scenarios, must be resolved by domain experts (experienced HAZOP conductors). This includes the definition of control instances that decide which results are correct, such as the HAZOP leader. Different roles of responsible parties such as creation, review, and clearance should be distinguished to comply with the duty of care when using knowledge-based systems.

The completeness of the automatically generated results depends directly on the input data object provided to the knowledge-based system. Analogous to preparing the HAZOP study, documents must be acquired to create the input data object. Accordingly, organizational measures must be taken to ensure the completeness of the input data object. After creation, the input data

object must be reviewed and approved. Once input data objects have been created, they can be stored, retrieved, and adapted during revisions, keeping the effort within limits. By versioning the input data object, the computer code of the knowledge-based system, the ontology, and the automatically generated worksheets in the form of HTML files, the influence of changes on the result can be immediately traced. In the management of change (MOC), versioning must be given particular importance.

7.4.4 Implications for HAZOP studies in the future

As technology advances in the field of natural speed recognition and computational linguistics, it seems reasonable that in the future, the task of scribing in HAZOP studies could be performed by a virtual assistant. Accordingly, the documentation process could be automated. The domain-specific ontology developed within this thesis could serve as a basis for recognizing HAZOP-specific relationships during the development of such a system since essential relationships between typical HAZOP scenarios and the dependencies to plants and substances are represented in machine-readable form.

In the case in the future, digital twins of process plants are available, and knowledge-based systems may be able to perform hazard identification and risk assessment automatically; process plants could be enabled to quantify the risk they pose themselves. The operations manager could have an online risk key performance indicator (R-KPI) on a dashboard. Thus, intelligent plants could be enabled to reduce the risk autonomously they pose to an acceptable level through specific countermeasures.

The use of immature technologies can give rise to new risks. If automated systems fail, make wrong recommendations, or make wrong decisions, humans must take over the automated functions or tasks. Because HAZOP analysis and follow-up studies and processes are of significant safety importance and, in the last instance, humans must make decisions about situations, the human expert must remain in the loop concerning (partially) automated processes and analyses.

7.5 Limitations of the developed methodology

The runtime of the inference algorithm was between 81.4-131.9 minutes for plant sections consisting of 4-5 equipment entities and leading to 55 to 63 scenarios for each plant section. One way to reduce the runtime is to optimize the number of reasoner calls by making the runs related to substances, deviations, and intended function or mode of operation less sequential and more simultaneous. Techniques such as multiprocessing or multithreading can reduce the knowledge-based system's calculation time for larger plant sections. Since the runtime of the propagation mode of the knowledge-based system increases with the complexity of the analyzed plant sections, the usage of multiprocessing or multithreading within the propagation mode should be investigated further. For instance, separate strands that merge or separate could be evaluated separately. Also, the handling of automatically generated instances should be investigated in more detail since the number of generated individuals impacts the runtime of the reasoner (cf. Table 32).

If new concepts become too specific, they may no longer be represented based on the available restrictions, including extended boundary conditions, leading to OWL 2 no longer being sufficient as an ontology language and SWRL having to be used. The advantage is that OWL 2 can be extended by SWRL rules. The Owlready2 package also contains the Pellet reasoner, which can process SWRL rules. However, it must be noted that the runtime of the pellet reasoner compared to the HermiT used within Owlready2 was larger by a factor of 1.5. This situation requires further investigation. Nevertheless, all concepts could be represented within this research work without SWRL using OWL 2.

Scenario components, scenario-dependent risk assessment, and proposal of safeguards were iteratively adjusted and evaluated to achieve the required quality of the HAZOP worksheets under

the aspects of *accuracy*, *completeness*, and *relevance*. If it is accepted that the developed knowledge-based system generates non-plausible or incomplete results, the developed approach can be directly applied to other processes and plants. It can be concluded that as soon as the knowledge-based system can process a specific technical system in the required quality, the computer system will also identify plausible scenarios for similar processes or plant sections. When the computer system is applied to entirely unknown processes and plant types, only those scenarios are identified which can be abstracted and derived from the concepts already modeled. Accordingly, it is essential to define specific plant types (typicals) or equipment configurations in connection with specific substances to extend the developed knowledge-based system successively. For each plant type and process added to the repertoire of the knowledge-based system, the potential ability to analyze new plant sections without making excessive adjustments increases.

Different valve positions in more complex plant sections lead to different topologies, leading to different SDGs. Accordingly, the analysis must be repeated for each SDG configuration, i.e., the SDG must be adapted, and the knowledge-based system must analyze the system again. Ergo, dynamic behavior is not automatically considered. However, this circumstance is not untypical for conventional HAZOP studies where different valve positions are considered for specific plant components in different operating modes. Besides, the algorithm for decomposing complex plant sections in basic plant sections and assembly it again during analysis requires refinement (cf. Section 5.14.1). The plants shown in Appendix Figure R-1 and Appendix Figure R-2 cannot be analyzed automatically in their entirety but must be divided into smaller sections, as shown in the case studies (cf. Sections 6.1.1 and 6.2.1).

The risk of scenarios reduced by safeguards is not automatically determined within the developed knowledge-based system. The selection of specific safeguards from the list of suggestions is done by the plant manager or the responsible project manager. The system could be extended to automatically determine the reduced risk after a manual selection of safeguards.

When considering the scenario component Consequence, the chain of events is interrupted after "RiskOfExplosiveAtmosphere" or "FireHazard", and subsequent consequences such as thermal radiation, pressure wave, flying debris, and domino effects are not considered. Quantitative consequence analyses are required to evaluate consequences and domino effects in a meaningful way which is outside the scope of this work.

The developed method and knowledge-based system currently reach its limits when detecting interactions between different substances in undesired chemical reactions and deriving plausible scenarios from them. Nevertheless, undesired polymerization or violent reactions with oxidizers are currently detected. Furthermore, only rudimentary scenario components are defined for chemical reactors (cf. Section 5.10.1). Concerning such scenarios, the functionality of the knowledge-based system can be extended by adding a reaction matrix and associated scenario components to the ontology. In addition, the case base for CBR may need to be extended for backward reasoning by further cases. The developed knowledge-based system reaches its limits when applied to process plants for which scenarios arise that are difficult to abstract or transfer. This includes scenarios that depend on specific information derived from process-specific details and information outside the system boundaries. In this case, no entirely plausible results can be generated with the required completeness. However, this does not mean that valid results are no longer generated; scenarios resulting from the structural configuration of the plants and scenarios that can be derived from general concepts are still valid.

8. Conclusion

Given the immense time and resources required and the team-specific quality of the achieved results of HAZOP studies, this research aimed to develop, implement, and investigate a knowledge-based system for digitizing the HAZOP method. As part of this work, the state of knowledge regarding automating HAZOP analysis approaches has been thoroughly reviewed and evaluated. More than 250 papers, book chapters, and online resources were analyzed in an extensive literature survey concerning past and contemporary research trends as well as methodological and technological aspects. In total, 32 approaches addressing aspects of HAZOP automation were analyzed and evaluated in detail. Based on these findings, research gaps were identified, and research questions were generated that could be answered within the scope of this dissertation. The following aspects were investigated in detail: structure of scenarios, causal relationships between scenario components, level of detail and abstraction of equipment, construction of overall scenarios, propagation of events, the topology of plant sections, risk assessment, and identification of safeguards. The result was a developed knowledge-based system that emulates the HAZOP method. For implementation, the programming language Python was used. The approach can be divided into three main components: (1) an ontology for representing expert knowledge, (2) a graph-based, object-oriented representation of plant sections, and (3) an inference algorithm for emulating the HAZOP method.

Knowledge acquisition was conducted by analyzing typical industry HAZOP worksheets, expert interviews, and technical literature. Given the diversity of disciplines, such as process engineering, process control, plant engineering, and safety engineering, the knowledge to be represented was complex and interdisciplinary. A critical aspect of this research was investigating how typical HAZOP scenarios can be deconstructed into components that form a sequence of events. A scenario model always reduces reality because only then can causal chains be established. For the modeling of scenarios, HAZOP concepts, such as cause, (process) deviation, and consequence, were extended by the concepts *underlying cause* and *effect*. Refining the causal chain provides the advantage that causal relationships between scenario components can be better modeled, and thus scenarios can be generated dynamically based on the scenario components. Dynamic modeling helps mitigate the *content completeness problem* and scalability issues compared to static modeling of scenarios.

In general, ontology-based knowledge representations offer the possibility to store interdisciplinary safety-related expert knowledge in a long-term and person-independent manner. Therefore, HAZOP scenarios were represented in the form of an ontology-based model. A curative approach was selected to create the ontology, meaning concepts were created manually based on aggregated expert knowledge. The developed ontology for knowledge representation was formalized using the state-of-the-art ontology language OWL 2. The implementation was done programmatically using the Python package Owlready2, which provided the advantages of applying best practices from the field of software development, such as version control, refactoring, and unit tests. The developed ontology was modularized to reuse non-application-specific components, including risk matrix, plant item taxonomy, and characterization of substances.

For the modeling of the plant sections to be analyzed, an SDG-based approach was selected wherein each node was represented by an equipment entity object, substance object, and additional boundary conditions. An overly detailed representation of the process and equipment-specific details does not add value for scenario identification. Meanwhile, an overly coarse level of detail prevents specific scenarios from being detected. A suitable level of abstraction and detail was identified during development to transfer scenarios to other processes and plant components based on structural and functional characteristics. In this work, a reusable and extendable object-oriented equipment entity library was developed for assembling process plants. This library was derived from a plant item taxonomy created. In addition, site information, boundary conditions,

and relevant substance properties necessary for reliably identifying and risk ranking scenarios were identified and implemented within the ontology-based model.

Based on the scenario components and the plant and substance representation, an inference algorithm was developed which composed sequence chains representing the scenario. The inference algorithm used the semantic reasoner HermiT for forward reasoning and a self-implemented case-based reasoner for backward reasoning to identify subsequent scenarios based on other scenarios. The aspects *deviations*, *substances*, and *intended functions* were evaluated for each piece of equipment, and scenarios were identified based on these aspects within the developed algorithm. In this process, risk ranking of the identified scenarios was automatically performed, and adequate safeguards were proposed. Evaluating the generated results of the first approach revealed that a purely equipment-based analysis is insufficient because interactions between equipment are not reflected, leading to missed scenarios. Accordingly, the inference algorithm was extended by a propagation-based mode. It enabled scenario components to be propagated through adjacent equipment, which provided causes for scenarios, especially in non-linear plant configurations. Topology-specific strategies were also investigated and implemented into the knowledge-based system to efficiently explore different plant section layouts. The result of the knowledge-based system was an automatically generated and formatted extended HAZOP worksheet. The scenarios were presented in detail, assigned a risk class, and adequate safeguards were proposed.

An evolutionary development approach was selected because the suitability of an ontology can only be evaluated in conclusion with an inference algorithm. The feature set was refined and redefined during the development because, after completing individual components and examining these, new requirements arose, and old requirements changed. The criteria *accuracy*, *completeness*, *relevance*, and *reliability* were used to assess the automatically generated HAZOP worksheets.

The results of this research show that ontology-based causation models are suitable for automatically generating high-quality HAZOP worksheets based on a digital representation of a process plant using the developed knowledge-based system. Within this research, worksheets were generated for plant types such as a hexane storage tank and the feed section of an olefin dimerization plant. Due to the careful representation of the causal relationships of the scenarios, and the association with the risk class of the overall scenario, proposals for safeguards could be reliably determined. The automatic risk assessment based on scenario components and the proposal of adequate safeguards for interrupting the event sequence of scenarios are new aspects compared to previous research approaches. Since the intention of the node under consideration is essential for identifying scenarios, it was considered under the aspects of intended function, the operation mode of equipment, and substance task. The developed method can also identify operability issues, which are often neglected in typical industry HAZOP studies due to time constraints.

The runtime of the knowledge-based system depends on the number of deviations, equipment, substances, intended functions, and operation modes. When applied to complex topologies of plant configurations, the number of scenarios identified in the propagation mode increases at the expense of runtime. The developed and investigated knowledge-based system can be classified with a technology readiness level (TRL) of 4 because basic principles have been observed (TRL 1), technology concepts have been formulated (TRL 2), an experimental proof of concept has been performed (TRL 3), and validation within a laboratory environment has occurred (TRL 4).

The worksheets are only as good as the representation of the plant section, modeled scenario components, and performance of the implemented inference algorithm that assembles the scenarios and determines risk. Thus, human errors that otherwise occur within HAZOP studies and lead to missed scenarios are shifted to creating the ontology, programming of the knowledge-based system, and creation of the input data object. Potential sources of error in generating the input data object must be limited by creating a management process, such as implementing a multi-stage approval process.

Literature

Aamodt, A. and Nygård, M. (1995), "Different roles and mutual dependencies of data, information and knowledge", *Data & Knowledge Engineering*, Vol. 16 No. 3, pp. 191–222.

Abburu, S. (2012), "A Survey on Ontology Reasoners and Comparison", *International Journal of Computer Applications*, Vol. 57 No. 17, pp. 33–39.

Al-Ajlan, A. (2015), "The Comparison between Forward and Backward Chaining", *International Journal of Machine Learning and Computing*, Vol. 5 No. 2, pp. 106–113.

Altabbakh, H.M. (2013), *Risk Analysis: Comparative Study of Various Techniques*, Missouri University of Science and Technology, available at: http://scholarsmine.mst.edu/doctoral_dissertationshttp://scholarsmine.mst.edu/doctoral_dissertations/2252.

Angele, J. and Lausen, G. (2004), "Ontologies in F-logic", in Staab, S. and Studer, R. (Eds.), *Handbook on Ontologies*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 29–50.

Antoniou, G., Franconi, E. and van Harmelen, F. (2005), "Introduction to Semantic Web Ontology Languages", in Eisinger, N. and Małuszyński, J. (Eds.), *Reasoning Web: First International Summer School 2005, Msida, Malta, July 25-29, 2005, Revised Lectures*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–21.

Ashraf, J., Chang, E., Hussain, O.K. and Hussain, F.K. (2015), "Ontology usage analysis in the ontology lifecycle: A state-of-the-art review", *Knowledge-Based Systems*, Elsevier B.V., Vol. 80, pp. 34–47.

Asim, M.N., Wasim, M., Usman, M., Khan, G., Mahmood, W. and Abbasi, H.M. (2018), "A survey of ontology learning techniques and applications", pp. 1–24.

AspenTech. (n.d.). "Aspen Technology", *Https://Www.Aspentech.Com/En/Products/Pages/Aspen-Batch-Modeler*, available at: https://www.aspentech.com/en/products/pages/aspen-batch-modeler (accessed 23 February 2020).

Baader, F., Horrocks, I. and Sattler, U. (2008), "Chapter 3 Description Logics", in van Harmelen, F., Lifschitz, V. and Porter, B.B.T.-F. of A.I. (Eds.), *Handbook of Knowledge Representation*, Vol. 3, Elsevier, pp. 135–179.

Bainbridge, L. (1983), "Ironies of automation", *Automatica*, Vol. 19 No. 6, pp. 775–779.

Baldwin, D. and Kasper, G.M. (1986), "Toward representing management-domain knowledge", *Decision Support Systems*, Vol. 2 No. 2, pp. 159–172.

Barr, A. and Feigenbaum, E.A. (1981), "Chapter III - Representation of Knowledge", in Barr, A. and Feigenbaum, E.A.B.T.-T.H. of A.I. (Eds.), *The Handbook of Artificial Intelligence*, Butterworth-Heinemann, pp. 141–222.

Batres, R. (2004), "An ontology approach to support HAZOP studies", *Proceedings of the Asian Pacific Confederation of Chemical Engineering*.

Batres, R. and Akmal, S. (2013), "A Formal Concept Analysis-Based Method for Developing Process Ontologies", *Journal of Chemical Engineering of Japan*, Vol. 46, pp. 396–406.

Batres, R., West, M., Leal, D., Price, D., Masaki, K., Shimada, Y., Fuchino, T., *et al.* (2005), "An upper ontology based on ISO 15926", *Computers and Chemical Engineering*, Vol. 20, pp. 1543–1548.

Baybutt, P. (2012), "What Risk Reduction Measures Should be Credited in Process Hazard Analysis?", *Process Safety Progress*, Vol. 31 No. 4, pp. 359–362.

Baybutt, P. (2015), "A critique of the Hazard and Operability (HAZOP) study", *Journal of Loss Prevention in the Process Industries*, Elsevier Ltd, Vol. 33, pp. 52–58.

Baybutt, P. (2019), "HAZOP Practices to Avoid: Consequence-Before-Cause Approach", available at: https://www.primatech.com/technical/pt-notes/190-hazop-practices-to-avoid-consequence-before-cause-approach (accessed 20 October 2020).

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F. and Stein, L.A. (2004), "OWL Web Ontology Language Reference", available at: https://www.w3.org/TR/owl-ref/#equivalentClass-def (accessed 19 September 2020).

Bingi, R. (1995), "A framework for the comparative analysis and evaluation of knowledge representation schemes", *Information Processing & Management*, Vol. 31 No. 2, pp. 233–247.

Black, P.E. (2014), "'hypergraph'", *Dictionary of Algorithms and Data Structures*, available at: https://xlinux.nist.gov/dads/HTML/hypergraph.html (accessed 8 March 2020).

Bloomfield, R.E. and Parisaca-Vargas, A. (2015), "Using Ontologies to Support Model-based Exploration of the Dependencies between Causes and Consequences of Hazards", *Proceedings of the 7th International Conference on Knowledge Engineering and Ontology Development*, Lisbon, Portugal, available at:https://doi.org/10.1109/CAMAD.2018.8514952.

Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler; Uli, *et al.* (2009), "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax", available at: https://www.w3.org/TR/2009/REC-owl2-syntax-20091027/ (accessed 16 August 2020).

Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler; Uli, *et al.* (2012), "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)", available at: https://www.w3.org/TR/owl2-syntax/#Object_Property_Axioms (accessed 16 August 2020).

Boonthum, N., Mulalee, U. and Srinophakun, T. (2014), "A systematic formulation for HAZOP analysis based on structural model", *Reliability Engineering and System Safety*, Elsevier, Vol. 121, pp. 152–163.

Brachman, R.J. and Levesque, H.J. (2004), *Knowledge Representation and Reasoning*, *Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers, available at:https://doi.org/10.1016/B978-1-55860-932-7.X5083-3.

Brachman, R.J. and Schmolze, J.G. (1985), "An overview of the KL-ONE Knowledge Representation System", *Cognitive Science*, Vol. 9 No. 2, pp. 171–216.

Bragatto, P., Monti, M., Giannini, F. and Ansaldi, S. (2007), "Exploiting process plant digital representation for risk analysis", *Journal of Loss Prevention in the Process Industries*, Vol. 20, pp. 69–78.

Buxmann, P. and Schmidt, H. (2019), *Künstliche Intelligenz*, Springer Gabler.

Cahyono, S.C. (2019), "Comparison of document similarity measurements in scientific writing using Jaro-Winkler Distance method and Paragraph Vector method", *IOP Conference Series: Materials Science and Engineering*, Vol. 662 No. 5, available at:https://doi.org/10.1088/1757-899X/662/5/052016.

Cardoso, J. (2007), "The semantic web vision: Where are we?", *IEEE Intelligent Systems*, Vol. 22 No. 5, pp. 84–88.

Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. and Wilkinson, K. (2004), "Jena: Implementing the Semantic Web Recommendations", *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, Association for Computing Machinery, New York, NY, USA, pp. 74–83.

Catino, C.A. and Ungar, L.H. (1995), "Model-based approach to automated hazard identification of chemical plants", *AIChE Journal*, John Wiley & Sons, Ltd, Vol. 41 No. 1, pp. 97–109.

CCPS. (1998), *Guidelines for Design Solutions for Process Equipment Failures*, *Guidelines for Design Solutions for Process Equipment Failures*, John Wiley & Sons, Ltd, available at:https://doi.org/10.1002/9780470935286.

CCPS. (2001), *Layer of Protection Analysis: Simplified Process Risk Assessment.*

CCPS. (2008a), "Glossary", *Guidelines for Hazard Evaluation Procedures*, pp. i–xxvii.

CCPS. (2008b), "Scenario-Based Hazard Evaluation Procedures", *Guidelines for Hazard Evaluation Procedures*, 3 April.

CCPS. (2008c), "Introduction to the Guidelines", *Guidelines for Hazard Evaluation Procedures*, pp. 15–33.

CCPS. (n.d.). "Safeguard | AIChE", *CCPS Process Safety Glossary*, available at: https://www.aiche.org/ccps/resources/glossary/process-safety-glossary/safeguard (accessed 13 July 2021a).

CCPS. (n.d.). "Loss Event | AIChE", *CCPS Process Safety Glossary*, available at: https://www.aiche.org/ccps/resources/glossary/process-safety-glossary/loss-event (accessed 18 July 2021b).

CCPS. (n.d.). "Scenario | AIChE", *CCPS Process Safety Glossary*, available at: https://www.aiche.org/ccps/resources/glossary/process-safety-glossary/scenario (accessed 19 July 2021c).

CCPS. (n.d.). "Atmospheric Storage Tank | AIChE", available at: https://www.aiche.org/ccps/resources/glossary/process-safety-glossary/atmospheric-storage-tank (accessed 27 June 2021d).

CCPS. (n.d.). "Loss of Primary Containment (LOPC) | AIChE", *CCPS Process Safety Glossary*, available at: https://www.aiche.org/ccps/resources/glossary/process-safety-glossary/loss-primary-containment-lopc (accessed 1 July 2021e).

Ceylan, I.I., Mendez, J. and Peñaloza, R. (2015), "The Bayesian ontology reasoner is born!", *CEUR Workshop Proceedings*, Vol. 1387 No. Grk, pp. 8–14.

Ceylan, İ.İ. and Peñaloza, R. (2017), "The Bayesian Ontology Language BEL", *Journal of Automated Reasoning*, Vol. 58 No. 1, pp. 67–95.

Chae, H. and Yoon, Y.H. (1994), "Safety Analysis Using an Expert System in Chemical Processes", *Korean Journal of Chemical Engineering*, Vol. 11 No. 3, pp. 153–161.

Chittaro, L., Guida, G., Tasso, C. and Toppano, E. (1993), "Functional and teleological knowledge in the multimodeling approach for reasoning about physical systems: a case study in diagnosis", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23 No. 6, pp. 1718–1751.

Clarke, I. (2000), "Integrated risk management", *Institution of Chemical Engineers Symposium Series*, No. 147, pp. 551–562.

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F. and Stein, L.A. (2001), "DAML+OIL Reference Description", available at: https://www.w3.org/TR/daml+oil-reference (accessed 12 April 2020).

Crawford, J., Farquhar, A. and Kuipers, B. (1990), "QPC: A Compiler from Physical Models into Qualitative Differential Equations", *Proceedings AAAI-90.*

Crawley, F. and Tyler, B. (2015a), "Chapter 3 - The HAZOP Study Method", in Crawley, F. and Tyler, B.B.T.-H.G. to B.P. (Third E. (Eds.), , Elsevier, pp. 10–12.

Crawley, F. and Tyler, B. (2015b), "Appendix 3 - An Illustration of HAZOP Study for a Continuous Operation", *HAZOP: Guide to Best Practice (Third Edition)*, Elsevier, pp. 101–121.

Crawley, F. and Tyler, B. (2015c), "Chapter 4 - The Detailed HAZOP Study Procedure", in Crawley, F. and Tyler, B.B.T.-H.G. to B.P. (Third E. (Eds.), , Elsevier, pp. 13–28.

Cui, L., Shu, Y., Wang, Z., Zhao, J., Qiu, T., Sun, W. and Wei, Z. (2012), "HASILT: An intelligent software platform for HAZOP, LOPA, SRS and SIL verification", *Reliability Engineering and System Safety*, Elsevier, Vol. 108, pp. 56–64.

Cui, L., Zhao, J., Qiu, T. and Chen, B. (2008), "Layered Digraph Model for HAZOP Analysis of Chemical Processes", *Process Safety Progress*, Vol. 27, pp. 293–305.

Cui, L., Zhao, J. and Zhang, R. (2010), "The integration of HAZOP expert system and piping and instrumentation diagrams", *Process Safety and Environmental Protection*, Institution of Chemical Engineers, Vol. 88 No. 5, pp. 327–334.

Daramola, O., Stålhane, T., Omoronyia, I. and Sindre, G. (2013), "Using Ontologies and Machine

Learning for Hazard Identification and Safety Analysis", in Maalej, W. and Thurimella, A.K. (Eds.), *Managing Requirements Knowledge*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 117–141.

Daramola, O., Stålhane, T., Sindre, G. and Omoronyia, I. (2011), "Enabling hazard identification from requirements and reuse-oriented HAZOP analysis", *2011 4th International Workshop on Managing Requirements Knowledge, MaRK'11 - Part of the 19th IEEE International Requirements Engineering Conference, RE'11*, No. October 2015, pp. 3–11.

Davis, R., Shrobe, H. and Szolovits, P. (1993), "What Is a Knowledge Representation ?", *AI Magazine*, Vol. 14 No. 1, pp. 17–33.

DHC. (2017), "Safety Data Sheet: n-hexane", 13 October.

Ding, Y. (2001), *IR and AI: The Role of Ontology*.

Drummond, N., Horridge, M., Dameron, O., Rector, A. and Wang, H. (2006), "A Practical Introduction to Protégé", *Session 1: Primitive Classes*.

Duhon, H. (2017), *HAZOPS Should Be Fun - The Stream-Based HAZOP*, Edition 1., Gibson Applied Engineering and Technology, Inc (GATE), Houston, TX.

Dunjó, J., Fthenakis, V.M., Darbra, R.M., Vílchez, J.A. and Arnaldos, J. (2011), "Conducting HAZOPs in continuous chemical processes: Part I. Criteria, tools and guidelines for selecting nodes", *Process Safety and Environmental Protection*, Vol. 89 No. 4, pp. 214–223.

Dutta, S. and Bonissone, P.P. (1993), "Integrating case- and rule-based reasoning", *International Journal of Approximate Reasoning*, Vol. 8 No. 3, pp. 163–203.

Elkin, P.L., Bailey, K.R., Ogren, P. V., Bauer, B.A. and Chute, C.G. (1999), "A randomized double-blind controlled trial of automated term dissection.", *Proceedings / AMIA ... Annual Symposium. AMIA Symposium*, pp. 62–66.

ErgoAI. (n.d.). "ErgoAI - Coherent Knowledge", available at: http://coherentknowledge.com/ergo-documentation/ (accessed 28 November 2020).

Faust, B. (2007), "Implementation of tacit knowledge preservation and transfer methods", *International Conference on Knowledge Management in Nuclear Facilities*, International Atomic Energy Agency (IAEA), Vienna, pp. 109–111.

Feigenbaum, E.A. (1982), *Knowledge Engineering in the 1980s*, Stanford.

Fernández, M., Gómez-Pérez, A. and Juristo, N. (1997), "METHONTOLOGY: From Ontological Art Towards Ontological Engineering".

Fikes, R. and Kehler, T. (1985), "The role of frame-based representation in reasoning", *Communications of the ACM*, Vol. 28 No. 9, pp. 904–920.

Folger, R. and Stein, C. (2017), "Abduction 101: Reasoning processes to aid discovery", *Human Resource Management Review*, Elsevier Inc., Vol. 27 No. 2, pp. 306–315.

Fuentes-Bargues, J.L., González-Cruz, M.C., González-Gaya, C. and Baixauli-Pérez, M.P. (2017), "Risk analysis of a fuel storage terminal using HAZOP and FTA", *International Journal of Environmental Research and Public Health*, Vol. 14 No. 7, available at:https://doi.org/10.3390/ijerph14070705.

Galluzzo, M., Bartolozzi, V. and Rinaudo, C. (1999), "Automating HAZOP analysis of batch processes", *Computers and Chemical Engineering*, Elsevier Science Ltd, Vol. 23 No. SUPPL. 1, pp. S661–S664.

Gao, D., Bai, Y. and Peng, L. (2020), "Reasoning Based on HAZOP Ontology", *Proceedings - 2020 Chinese Automation Congress, CAC 2020*, pp. 6170–6174.

Genesereth, M.R. and Fikes, R.E. (1992), *Knowledge Interchange Format*, Stanford,California.

Giarratano, J.C. and Riley, G.D. (2005), *Expert Systems: Principles and Programming*, Brooks/Cole Publishing Co., USA.

Glimm, B., Horrocks, I., Motik, B. and Stoilos, G. (2010), "Optimising Ontology Classification", in Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., et al. (Eds.), *The Semantic Web – ISWC 2010*, Springer Berlin Heidelberg, Berlin, Heidelberg,

pp. 225–240.

Gonçalves, R., Matentzoglu, N., Parsia, B. and Sattler, U. (2013), "The Empirical Robustness of Description Logic Classification", in Eiter, T., Glimm, B., Kazakov, Y. and Krötzsch, M. (Eds.), *Informal Proceedings of the 26th International Workshop on Description Logics*, Ulm, pp. 197–208.

Göring, M.H. and Schecker, H.G. (1993), "HAZEXPERT — an integrated expert system to support hazard analysis in process plant design", *Computers & Chemical Engineering*, Vol. 17, pp. 429–434.

Graf, H. and Schmidt-Traub, H. (2000), "Early hazard identification of chemical plants with statechart modelling techniques", *Safety Science*, Vol. 36 No. 1, pp. 49–67.

Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. and Sattler, U. (2008), "OWL 2: The next step for OWL", *Journal of Web Semantics*, Vol. 6 No. 4, pp. 309–322.

Grimm, S., Hitzler, P. and Abdecker, A. (2007), "Knowledge Representation and Ontologies", in Studer, R., Grimm, S. and Abecker, A. (Eds.), *Semantic Web Services: Concepts, Technologies, and Applications*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 51–105.

Gruber, T. (1995), "Toward principles for the desing of ontologies used for knowledge sharing", *Academic Press Limited*.

Gruber, T.R. (1993), "A translation approach to portable ontology specifications", *Knowledge Acquisition*, Vol. 5, pp. 199–220.

Guarino, N. (1998), "Formal Ontology and Information Systems", *Formal Ontology in Information Systems: Proceedings of the 1st International Conference*, Vol. 46 No. June, pp. 3–15.

Haarslev, V., Hidde, K., Möller, R. and Wessel, M. (2012), "The RacerPro knowledge representation and reasoning system", *Semantic Web*, Vol. 3 No. 3, pp. 267–277.

Hadikusumo, B. and Rowlinson, S. (2004), "Capturing Safety Knowledge Using Design-for-Safety-Process Tool", *Journal of Construction Engineering and Management*, Vol. 130, available at:https://doi.org/10.1061/(ASCE)0733-9364(2004)130:2(281).

Hagberg, A.A., Schult, D.A. and Swart, P.J. (2008), "Exploring network structure, dynamics, and function using NetworkX", in Varoquaux, G., Vaught, T. and Millman, J. (Eds.), *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15.

Harel, D. (1988), "On Visual Formalisms", *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, Vol. 31 No. 5, pp. 514–530.

Harmelen, F., Lifschitz, V. and Porter, B. (2007), "The Handbook of Knowledge Representation", *Elsevier Science San Diego, USA*, p. 1034.

Heino, P. (1998), *Fluid Property Reasoning in Knowledge-Based Hazard Identification*, Loughborough.

Heino, P., Kotikunnas, E., Shei, W.F., Shao, C.C. and Chen, C.H. (1995), "Computer-aided HAZOP with knowledge-based identification of hazardous event chains", *Loss Prevention and Safety Promotion in the Process Industries*, Elsevier, Vol. 1, pp. 645–656.

Heino, P., Poucet, A. and Suokas, J. (1992), "Computer tools for hazard identification, modelling and analysis", *Journal of Hazardous Materials*, Vol. 29 No. 3, pp. 445–463.

Hitzler, P., Kroetzsch, M., Parsia, B. and Patel-Schneider, Peter F. Rudolph, S. (2012), "OWL 2 Web Ontology Language Primer (Second Edition)", available at: https://www.w3.org/TR/owl2-primer/#Datatypes (accessed 20 September 2020).

Hitzler, P., Krötzsch, M. and Rudolph, S. (2009), "Knowledge Representation for the Semantic Web Part II: Rules for OWL".

Hitzler, P., Krötzsch, M. and Rudolph, S. (2010), *Foundations of Semantic Web Technologies*, CRC Press.

Hitzler, P., Krötzsch, M., Rudolph, S. and Sure, Y. (2008), *Semantic Web*, 1st ed., Springer.

Hoekstra, R. (2009), *Ontology Representation: Design Patterns and Ontologies That Make Sense*, University of Amsterdam, available at:https://doi.org/10.3233/978-1-60750-013-1-i.

Hofinger, G. (2008), "Fehler und Unfälle", in Badke-Schaub, P., Hofinger, G. and Lauche, K. (Eds.), *Human Factors: Psychologie Sicheren Handelns in Risikobranchen*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 36–55.

Hofmann, F. (2018), "Levenshtein Distance and Text Similarity in Python", available at: https://stackabuse.com/levenshtein-distance-and-text-similarity-in-python/ (accessed 21 March 2020).

Hollnagel, E. (2012), *FRAM: The Functional Resonance Analysis Method: Modelling Complex Socio-Technical Systems*, available at:https://doi.org/10.1201/9781315255071.

Horridge, M. (2011), *A Practical Guide To Building OWL OntologiesUsing Protégé 4 and CO-ODE ToolsEdition 1.3*, Manchester.

Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R. and Wang, H.H. (2006), "The manchester OWL syntax", in Grau, B.C., Hitzler, P., Shankey, C. and Wallace; Evan (Eds.), *CEUR Workshop Proceedings*, Vol. 216, Athens, Georgia (USA), available at: CEUR-WS.org.

Horrocks, I. (2005), "OWL: A Description Logic Based Ontology Language", in van Beek, P. (Ed.), *Principles and Practice of Constraint Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 5–8.

Horrocks, I. (2008), "Ontologies and the semantic web", *Communications of the ACM*, Vol. 51 No. 12, p. 58.

Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B. and Dean, M. (2004), "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", available at: https://www.w3.org/Submission/SWRL/ (accessed 11 April 2020).

Hyatt, N. (2003), *Guidelines for Process Hazards Analysis (PHA, HAZOP), Hazards Identification, and Risk Analysis*, *Guidelines for Process Hazards Analysis (PHA, HAZOP), Hazards Identification, and Risk Analysis*, available at:https://doi.org/10.1201/9781420039603.

Ichikawa, J.J. and Steup, M. (2018), "The Analysis of Knowledge", in Zalta, E.N. (Ed.), *The Stanford Encyclopedia of Philosophy*, Summer 201., Metaphysics Research Lab, Stanford University, available at: https://plato.stanford.edu/archives/sum2018/entries/knowledge-analysis/.

IVSS. (2020), *Das PAAG- / HAZOP-Verfahren Und Weitere Praxisbewährte Methoden*.

Janowski, R. (1987), "An introduction to QSIM and qualitative simulation", *Artificial Intelligence in Engineering*, Vol. 2 No. 2, pp. 65–71.

Johnson-Laird, P.N. (1999), "Deductive reasoning", *Annual Review of Psychology*, Vol. 50, pp. 109–135.

Kalibatiene, D. and Vasilecas, O. (2011), *Survey on Ontology Languages*, Vol. 90, available at:https://doi.org/10.1007/978-3-642-24511-4.

Kang, B., En Sup Yoon and Jung Chul Suh. (2001), "Application of automated hazard analysis by new multiple process-representation models to chemical plants", *Industrial and Engineering Chemistry Research*, Vol. 40 No. 8, pp. 1891–1902.

Kang, B., Lee, B., Kang, K.W., Suh, J.C. and Yoon, E.S. (1999), "AHA: A knowledge based system for automatic hazard identification in chemical plant by multimodel approach", *Expert Systems with Applications*, Vol. 16 No. 2, pp. 183–195.

Kappelmaier, R., Poschenreithner, G., Rossinelli, L., Sommer, J. and Uhlmann, G. (2008), *Das PAAG-Verfahren*, 4th ed., Internationale Sektion der IVSS für die Verhütung von Arbeitsunfällen und Berufskrankheiten in der chemischen Industrie, Heidelberg.

Kern-Isberner, G., Douven, I., Knauff, M. and Prade, H. (2017), "Reasoning with Imperfect Information and Knowledge", *Minds and Machines*, Springer Netherlands, Vol. 27 No. 1, pp. 7–9.

Khan, F., Rathnayaka, S. and Ahmed, S. (2015), "Methods and models in process safety and risk

management : Past , present and future", *Process Safety and Environmental Protection*, Institution of Chemical Engineers, Vol. 98, pp. 116–147.

Khan, F.I. and Abbasi, S.A. (1997a), "TOPHAZOP: A knowledge-based software tool for conducting HAZOP in a rapid, efficient yet inexpensive manner", *Journal of Loss Prevention in the Process Industries*, Vol. 10 No. 5–6, pp. 333–343.

Khan, F.I. and Abbasi, S.A. (1997b), "OptHAZOP - An effective and optimum approach for HAZOP study", *Journal of Loss Prevention in the Process Industries*, Vol. 10 No. 3, pp. 191–204.

Khan, F.I. and Abbasi, S.A. (1997c), "Mathematical model for HAZOP study time estimation", *Journal of Loss Prevention in the Process Industries*, Vol. 10 No. 4, pp. 249–257.

Khan, F.I. and Abbasi, S.A. (2000), "Towards automation of HAZOP with a new tool EXPERTOP", *Environmental Modelling and Software*, Vol. 15 No. 1, pp. 67–77.

Kidam, K. and Hurme, M. (2013), "Analysis of equipment failures as contributors to chemical process accidents", *Process Safety and Environmental Protection*, Institution of Chemical Engineers, Vol. 91 No. 1–2, pp. 61–78.

Kifer, M. and Lausen, G. (1989), "F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme", *ACM SIGMOD Record*, Vol. 18 No. 2, pp. 134–146.

Kletz, T.A. (1997), "HAZOP - Past and future", *Reliability Engineering and System Safety*, Vol. 55 No. 3, pp. 263–266.

Kletz, T.A. (1999), *Hazop and Hazan: Identifying and Assessing Process Industry Hazards*, Institution of Chemical Engineers, available at: https://books.google.de/books?id=m2u577x_6U4C.

Kotek, L. and Tabas, M. (2012), "HAZOP study with qualitative risk analysis for prioritization of corrective and preventive actions", *Procedia Engineering*, Vol. 42 No. August, pp. 808–815.

Kraus, A. and Schecker, H.G. (2002), "Wissensbasierte systeme zur identifikation und beurteilung von gefahren und risiken in prozessanlagen", *Chemie-Ingenieur-Technik*, Vol. 74 No. 1–2, pp. 121–125.

Krötzsch, M., Simančík, F. and Horrocks, I. (2014), "A description logic primer", *Perspectives on Ontology Learning*, Vol. 18 No. June, pp. 3–20.

Kuipers, B. (1987), "Qualitative Simulation as Causal Explanation", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, pp. 432–444.

Lamy, J.B. (2017), "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies", *Artificial Intelligence in Medicine*, Elsevier B.V., Vol. 80, pp. 11–28.

Lantow, B. (2016), "OntoMetrics : Application of on-line Ontology Metric Calculation", in Johansson, B. and Vencovský, F. (Eds.), *CEUR Workshop Proceedings*, Prague, pp. 1–12.

Lassila, O. and Swick, R.R. (1997), "Resource Description Framework (RDF) Model and Syntax", available at: https://www.w3.org/TR/WD-rdf-syntax-971002/ (accessed 11 April 2020).

Latombe, J.-C. (1987), "The Role of Reasoning in Knowledge-Based Systems BT - Wissensbasierte Systeme", in Brauer, W. and Wahlster, W. (Eds.), , Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 159–167.

Lehmann, F. (1992), "Semantic networks", *Computers & Mathematics with Applications*, Vol. 23 No. 2, pp. 1–50.

Lehto, M. and Salvendy, G. (1991), "Models of accident causation and their application: Review and reappraisal", *Journal of Engineering and Technology Management*, Vol. 8 No. 2, pp. 173–205.

Leveson, N.G. (2019), *Shortcomings of the Bow Tie and Other Safety Tools Based on Linear Causality*, available at: http://sunnyday.mit.edu/Bow-tie-final.pdf.

Leveson, N.G. and Stephanopoulos, G. (2014), "A system-theoretic, control-inspired view and approach to process safety", *AIChE Journal*, Vol. 60 No. 1, pp. 2–14.

Leveson, N.G. and Thomas, J.P. (2018), *STPA Handbook*, available at: http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

Levin, M.K., Ruttenberg, A., Masci, A.M. and Cowell, L.G. (2015), "Owl-cpp, a C++ library for working with OWL ontologies", *Journal of Biomedical Semantics*, Journal of Biomedical Semantics, Vol. 6 No. 35, available at:https://doi.org/10.1186/s13326-015-0035-z.

Liebig, T. (2006), *Reasoning with OWL: System Support and Insights*, Ulm, available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Reasoning+with+OWL+-+System+Support+and+Insights+-#0.

Lifschitz, V., Morgenstern, L. and Plaisted, D. (2008), "Chapter 1 Knowledge Representation and Classical Logic", in van Harmelen, F., Lifschitz, V. and Porter, B.B.T.-F. of A.I. (Eds.), *Handbook of Knowledge Representation*, Vol. 3, Elsevier, pp. 3–88.

Liin Rossing, N., Lind, M., Jensen, N. and Bay Jørgensen, S. (2009), "A Goal Based HAZOP Assistant", *Computer Aided Chemical Engineering*, Vol. 26, pp. 1129–1134.

Lind, M. (1981), "The use of Flow Models for Automated Plant Diagnosis", in Rasmussen, J. and Rouse, W.B. (Eds.), *Human Detection and Diagnosis of System Failures*, Springer US, Boston, MA, pp. 411–432.

Lind, M. (2004), *MODELING GOALS AND FUNCTIONS OF COMPLEX PLANT*, Lyngby.

Lind, M. (2011), "An introduction to multilevel flow modeling", *International Journal of Nuclear Safety and Simulation*, Vol. 2, pp. 22–32.

List, R. (2017), "Einführung in CATIA V5 BT - CATIA V5 – Grundkurs für Maschinenbauer: Bauteil- und Baugruppenkonstruktion, Zeichnungsableitung", in List, R. (Ed.), , Springer Fachmedien Wiesbaden, Wiesbaden, pp. 33–40.

Lü, N. and Wang, X. (2007), "SDG-Based HAZOP and Fault Diagnosis Analysis to the Inversion of Synthetic Ammonia", *Tsinghua Science and Technology*, Vol. 12 No. 1, pp. 30–37.

MacGregor, R.J. (2013), "Assess hazards with process flow failure modes analysis", *Chemical Engineering Progress*, Vol. 109 No. 3, pp. 48–56.

Maedche, A. and Staab, S. (2001), "Ontology Learning for the Semantic Web", *IEEE Intelligent Systems*, Vol. 16 No. 2, pp. 72–79.

Mannan, S. (2012), "Chapter 8 - Hazard Identification", in Mannan, S.B.T.-L.L.P. in the P.I. (Fourth E. (Ed.), *Lees' Loss Prevention in the Process Industries*, 4th ed., Butterworth-Heinemann, Oxford, pp. 204–283.

Manzey, D. (2012), "Systemgestaltung und Automatisierung", in Badke-Schaub, P., Hofinger, G. and Lauche, K. (Eds.), *Human Factors: Psychologie Sicheren Handelns in Risikobranchen*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 333–352.

Mao, S., Zhao, Y., Chen, J., Wang, B. and Tang, Y. (2020), "Development of process safety knowledge graph: A Case study on delayed coking process", *Computers and Chemical Engineering*, Elsevier Ltd, Vol. 143, p. 107094.

Maragakis, I., Clark, S., Piers, M., Prior, D., Tripaldi, C., Masson, M. and Audard, C. (2009), "Guidance on Hazard Identification", *Safety Management System and Safety Culture Working Group (SMS WG)*, available at: https://www.easa.europa.eu/sites/default/files/dfu/ECASTSMSWG-GuidanceonHazardIdentification1.pdf.

Marchetti, A., Ronzano, F., Tesconi, M. and Minutoli, S. (2008), "Formalizing Knowledge by Ontologies: OWL and KIF", p. 19.

Marhavilas, P.K., Filippidis, M., Koulinas, G.K. and Koulouriotis, D.E. (2020), "A HAZOP with MCDM based risk-assessment approach: Focusing on the deviations with economic/health/environmental impacts in a process industry", *Sustainability (Switzerland)*, Vol. 12 No. 3, available at:https://doi.org/10.3390/su12030993.

Marx, M. (2003), *Methodenentwicklung Zur Verknüpfung von Sicherheitsuntersuchungen Und CAD*, Otto-von-Guericke-Universität Magdeburg.

McBride, B. (2004), "RDF Vocabulary Description Language 1.0: RDF Schema", available at: https://www.w3.org/2001/sw/RDFCore/TR/WD-rdf-schema-20030117/ (accessed 12 April 2020).

McBride, M., Marsh, C., Herbert, I. and Robinson, C. (2009), "Retrospective hazard identification and assessment of ageing plant", *Institution of Chemical Engineers Symposium Series*, No. 155, pp. 468–476.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Chung, P.W.H., Rushton, A.G. and Lees, F.P. (2000), "HAZID, a computer aid for hazard identification: 4. Learning set, main study system, output quality and validation trials", *Process Safety and Environmental Protection*, Vol. 78 No. 2, pp. 91–119.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Chung, P.W.H., Rushton, A.G., Lees, F.P. and Heino, P.M. (1999), "HAZID, a computer aid for hazard identification: 3. The fluid model and consequence evaluation systems", *Process Safety and Environmental Protection*, The Institution of Chemical Engineers, Vol. 77 No. 6, pp. 335–353.

McCoy, S.A., Wakeman, S.J., Larkin, F.D., Jefferson, M.L., Chung, P.W.H., Rushton, A.G., Lees, F.P., *et al.* (1999), "HAZID, a computer aid for hazard identification: 1. The STOPHAZ package and the HAZID code: An overview, the issues and the structure", *Process Safety and Environmental Protection*, The Institution of Chemical Engineers, Vol. 77 No. 6, pp. 317–327.

McDermid, J.A., Nicholson, M., Pumfrey, D.J. and Fenelon, P. (1995), "Experience with the application of HAZOP to computer-based systems", *COMPASS "95 Proceedings of the Tenth Annual Conference on Computer Assurance Systems Integrity, Software Safety and Process Security"*, IEEE, pp. 37–48.

Mechhoud, E.A., Rouainia, M. and Rodriguez, M. (2016), "A new tool for risk analysis and assessment in petrochemical plants", *Alexandria Engineering Journal*, Faculty of Engineering, Alexandria University, Vol. 55 No. 3, pp. 2919–2931.

Minsky, M. (1974), *A Framework for Representing Knowledge*, Massachusetts Institute of Technology, USA.

Mock, R. and van Mahnen, J. (1998), "Risk Analysis Methods in Processing Industries", *Proceedings of the Society for Risk Analysis-Europe (SRA-E)*, Paris, pp. 1145–1156.

Mohammadfam, I., Kalatpour, O., Golmohammadi, R. and Khotanlou, H. (2013), "Developing a process equipment failure knowledge base using ontology approach for process equipment related incident investigations", *Journal of Loss Prevention in the Process Industries*, Elsevier Ltd, Vol. 26 No. 6, pp. 1300–1307.

Morbach, J., Bayer, B., Wiesner, A., Yang, A. and Marquardt, W. (2008), "OntoCAPE 2.0. The Upper level", Vol. 49 No. 0, p. 85.

Morbach, J., Wiesner, A. and Marquardt, W. (2009), "OntoCAPE-A (re)usable ontology for computer-aided process engineering", *Computers and Chemical Engineering*, Vol. 33 No. 10, pp. 1546–1556.

Mosier, K.L. and Skitka, L.J. (1996), "Human decision makers and automated decision aids: Made for each other?", *Automation and Human Performance: Theory and Applications*, pp. 201–220.

Munir, K. and Sheraz Anjum, M. (2018), "The use of ontologies for effective knowledge modelling and information retrieval", *Applied Computing and Informatics*, King Saud University, Vol. 14 No. 2, pp. 116–126.

Musen, M.A. (2015), "The protégé project: a look back and a look forward", *AI Matters*, Vol. 1 No. 4, pp. 4–12.

Nalepa, G.J. (2017), *Modeling with Rules Using Semantic Knowledge Engineering*, 1st ed., Springer International Publishing.

Neo4j. (n.d.). "Graph Database Platform | Graph Database Management System | Neo4j",

available at: https://neo4j.com/ (accessed 25 August 2021).

Newell, A. and Simon, H.A. (1972), *Human Problem Solving*, Prentice-Hall, available at: https://books.google.de/books?id=h03uAAAAMAAJ.

Nolan, D. (2015), "Appendix E - HAZOP Parameters, Deviations, and Possible Causes", in Nolan, D.P.B.T.-S. and S.R. for the P.I. (Fourth E. (Ed.), *Safety and Security Review for the Process Industries*, Gulf Publishing Company, pp. 141–150.

Noy, N.F. and McGuinness, D.L. (n.d.). *Ontology Development 101: A Guide to Creating Your First Ontology*, Stanford, available at: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf (accessed 6 August 2020).

OREDA. (2002), *OREDA : Offshore Reliability Data Handbook.*, OREDA Participants : Distributed by Der Norske Veritas, HØvik, Norway.

Othman, M.R., Idris, R., Hassim, M.H. and Ibrahim, W.H.W. (2016), "Prioritizing HAZOP analysis using analytic hierarchy process (AHP)", *Clean Technologies and Environmental Policy*, Springer Berlin Heidelberg, Vol. 18 No. 5, pp. 1345–1360.

Oyeleye, O.O. and Kramer, M.A. (1987), "Qualitative Simulation of Process Plants", *IFAC Proceedings Volumes*, Elsevier, Vol. 20 No. 5, pp. 329–334.

Palmer, C. (1999), *Creating Signed Directed Graph Models for Process Plants*, Loughborough University, available at:https://doi.org/10.1021/ie990637v.

Parasuraman, R., Sheridan, T.B. and Wickens, C.D. (2000), "A Model for Types and Levels of Human Interaction with Automation", *IEEE Transactions on Systems Man and Cybernetics - Part A Systems and Humans*, Vol. 30 No. 3, pp. 286–297.

Park, S. (2017), *DATA-BASED SEMI-AUTOMATIC HAZARD IDENTIFICATION FOR MORE COMPREHENSIVE IDENTIFICATION OF HAZARDOUS SCENARIOS*, Texas A&M University.

Parmar, J.C. and Lees, F.P. (1987a), "The propagation of faults in process plants: Hazard identification for a water separator system", *Reliability Engineering*, Vol. 17 No. 4, pp. 303–314.

Parmar, J.C. and Lees, F.P. (1987b), "The propagation of faults in process plants: Hazard identification", *Reliability Engineering*, Vol. 17 No. 4, pp. 277–302.

Pasman, H.J. and Rogers, W.J. (2016), "How can we improve HAZOP, our old work horse, and do more with its results? an overview of recent developments", *Chemical Engineering Transactions*, Vol. 48, pp. 829–834.

Peng, Y., Jiang, Y. and Radivojac, P. (2018), "Enumerating consistent sub-graphs of directed acyclic graphs: An insight into biomedical ontologies", *Bioinformatics*, Vol. 34 No. 13, pp. i313–i322.

Peterson, J.L. (1981), *Petri Net Theory and the Modeling of Systems*, Prentice Hall PTR, USA.

Prentzas, J. and Hatzilygeroudis, I. (2007), "Categorizing approaches combining rule-based and case-based reasoning", *Expert Systems*, John Wiley & Sons, Ltd, Vol. 24 No. 2, pp. 97–122.

Primatech. (n.d.). "PHAWorks", available at: https://www.primatech.com/software/phaworks (accessed 25 August 2020).

Quillian, M. (1968), "Semantic Memory", in Minsky, M. (Ed.), *Semantic Information Processing*, MIT Press, pp. 227–270.

Rahman, S., Khan, F., Veitch, B. and Amyotte, P. (2009), "ExpHAZOP+: Knowledge-based expert system to conduct automated HAZOP analysis", *Journal of Loss Prevention in the Process Industries*, Elsevier Ltd, Vol. 22 No. 4, pp. 373–380.

Rasmussen, J. (1985), "The Role of Hierarchical Knowledge Representation in Decision making and System Management", *IEEE Transactions on Systems, Man and Cybernetics*, IEEE, Vol. SMC-15 No. 2, pp. 234–243.

Rasmussen, J. and Svedung, I. (2000), *Proactive Risk Management in a Dynamic Society*, Swedish

Rescue Services Agency, Karlstad, available at: http://rib.msb.se/Filer/pdf%5C16252.pdf.
Redmill, F., Chudleigh, M. and Catmur, J. (2000), "System Safety: HAZOP and Software HAZOP", edited by Morris, C.*Industrial Management & Data Systems*, Emerald Group Publishing Limited, Vol. 100 No. 1, pp. 46–48.
Roche, E., Dupont, W. and Summers, A. (2019), "Beyond HAZOP: Analyzing common cause and system scenarios", *Process Safety Progress*, Vol. 38 No. 2, pp. 1–5.
Rodríguez, M. and De la Mata, J.L. (2012), "Automating HAZOP studies using D-higraphs", *Computers and Chemical Engineering*, Elsevier Ltd, Vol. 45, pp. 102–113.
Rodríguez, M. and Laguía, J. (2019), "An ontology for process safety", *Chemical Engineering Transactions*, Vol. 77 No. May, pp. 67–72.
Rootsaert, T. and Harrington, J. (1992), *A KNOWLEDGE BASED COMPUTER PROGRAM "COMHAZOP" AS AID FOR HAZARD AND OPERABILITY STUDIES IN PROCESS PLANTS*, Terneuzen.
Rossing, N., Lind, M., Jensen, N. and Jørgensen, S.B. (2010), "A functional HAZOP methodology", *Computers and Chemical Engineering*, Vol. 34 No. 2, pp. 244–253.
Rousseau, D., Billingham, J. and Calvo-Amodio, J. (2018), "Systemic Semantics: A Systems Approach to Building Ontologies and Concept Maps", *Systems*, Vol. 6 No. 3, p. 32.
Rudolph, S. (2011), "Foundations of Description Logics", in Polleres, A., D'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S. and Patel-Schneider, P. (Eds.), *Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 76–136.
Schacter, D.L. (1994), "Implicit Knowledge: New Perspectives On Unconscious Processes", *International Review of Neurobiology*, Vol. 37 No. C, pp. 271–284.
Schaub, H. (2008), "Wahrnehmung, Aufmerksamkeit und „Situation Awareness" (SA)", in Badke-Schaub, P., Hofinger, G. and Lauche, K. (Eds.), *Human Factors: Psychologie Sicheren Handelns in Risikobranchen*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 59–76.
Schmalenbach, H.H. (2013), *Ontologien Zum Bereitstellen von Gestaltungswissen Am Beispiel von Ingenieurkeramik Ontologies*, Karlsruhe Institute of Technology.
Schmidberger, T., Scherf, T. and Fay, A. (2009), "Knowledge-based support of HAZOP studies using a CAEX plant model", *Inside Functional Safety*, pp. 5–15.
Schmolze, J.G. and Lipkis, T.A. (1983), "Classification in the KL-ONE Knowledge Representation System", *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 330–332.
Schneider, M., Rudolph, S. and Sutcliffe, G. (2013), "Modeling in OWL 2 without restrictions", *CEUR Workshop Proceedings*, Vol. 1080.
Schubach, S. (1997), "A modified computer hazard and operability study procedure", *Journal of Loss Prevention in the Process Industries*, Vol. 10 No. 5–6, pp. 303–307.
Schwarz, H. V., Koerts, T. and Hörcher, U. (2019), "Semiquantitative risk analysis – An EPSC working group", *Chemical Engineering Transactions*, Vol. 77 No. April, pp. 37–42.
Seligmann, B.J. (2011), *A Functional Systems Framework and Blended Hazard Identification Methodology to Support Process Diagnosis*, The University of Queensland.
Seligmann, B.J., Németh, E., Hangos, K.M. and Cameron, I.T. (2012), "A blended hazard identification methodology to support process diagnosis", *Journal of Loss Prevention in the Process Industries*, Elsevier Ltd, Vol. 25 No. 4, pp. 746–759.
Semafora. (n.d.). "OntoBroker and OntoStudio X | semafora semantic AI", available at: https://www.semafora-systems.com/ontobroker-and-ontostudio-x (accessed 28 November 2020).
Sheridan, T.B. (1992), *Telerobotics, Automation, and Human Supervisory Control*, MIT Press, Cambridge, MA, USA.
Shum, S.K., Davis, J.F., Punch, W.F. and Chandrasekaran, B. (1988), "An expert system

approach to malfunction diagnosis in chemical plants", *Computers & Chemical Engineering*, Vol. 12 No. 1, pp. 27–36.

Single, J.I., Schmidt, J. and Denecke, J. (2019a), "State of research on the automation of HAZOP studies", *Journal of Loss Prevention in the Process Industries*, Elsevier, Vol. 62 No. July, p. 103952.

Single, J.I., Schmidt, J. and Denecke, J. (2019b), "COMPUTER-AIDED HAZOP STUDIES: KNOWLEDGE REPRESENTATION AND ALGORITHMIC HAZARD IDENTIFICATION", *WIT Transactions on The Built Environment*, Vol. 189, pp. 55–66.

Single, J.I., Schmidt, J. and Denecke, J. (2020a), "Ontology-Based Support for Hazard and Operability Studies", *International Journal of Safety and Security Engineering*, Vol. 10 No. 3, pp. 311–319.

Single, J.I., Schmidt, J. and Denecke, J. (2020b), "Ontology-based computer aid for the automation of HAZOP studies", *Journal of Loss Prevention in the Process Industries*, Elsevier Ltd, Vol. 68 No. October, p. 104321.

Single, J.I., Schmidt, J. and Denecke, J. (2020c), "Computer-Aided Hazop : Ontologies and AI for Hazard Identification and Propagation", in Pierucci, S., Manenti, F., Bozzano, G. and Manca, D. (Eds.), *PROCEEDINGS OF THE 30th European Symposium on Computer Aided Process Engineering(ESCAPE30)*, Elsevier B.V., Milano, Italy, pp. 1783–1788.

Single, J.I., Schmidt, J. and Denecke, J. (2020d), "Knowledge acquisition from chemical accident databases using an ontology-based method and natural language processing", *Safety Science*, Vol. 129, p. 104747.

Sirin, E., Parsia, B., Grau, B., Kalyanpur, A. and Katz, Y. (2007), "Pellet: a practical OWL-DL reasoner", *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 5, pp. 51–53.

Sowa, J.F. (n.d.). "Conceptual Graph Summary", available at: http://www.jfsowa.com/cg/cgif.htm (accessed 11 April 2020).

Sphera. (n.d.). "Process Hazard Analysis (PHA) & HAZOP Software | Sphera", *Https://Sphera.Com/Operational-Risk/Pha-Pro/*, available at: https://sphera.com/operational-risk/pha-pro/ (accessed 23 February 2020).

Srinivasan, R. and Venkatasubramanian, V. (1996), "PETRI NET-DIGRAPH MODELS FOR AUTOMATING HAZOP ANALYSIS OF BATCH PROCESS PLANTS", *COMPUTERS AND*, Vol. 20 No. 96, pp. 719–725.

Srinivasan, R. and Venkatasubramanian, V. (1998), "Automating HAZOP analysis of batch chemical plants: Part I. The knowledge representation framework", *Computers and Chemical Engineering*, Vol. 22 No. 9, pp. 1345–1355.

Stevens, R. and Sattler, U. (2013), "An object lesson in choosing between a class and an object", *Ontogenesis*, 16 August, available at: http://ontogenesis.knowledgeblog.org/1418 (accessed 8 September 2021).

Stuckenschmidt, H. (2008), *Ontologien - Konzepte, Technologien Und Anwendungen*, *Berlin Und Heidelberg*, Springer-Verlag Berlin Heidelberg, available at:https://doi.org/10.1007/978-3-642-11460-1.

Styczynski, Z.A., Rudion, K. and Naumann, A. (2017), *Einführung in Expertensysteme*, Springer Vieweg, available at:https://doi.org/10.1007/978-3-662-53172-3.

Suh, J.C., Lee, B., Kang, I. and Yoon, E. (1997), "An expert system for automated hazard analysis based on multimodel approach", *Computers & Chemical Engineering*, Vol. 21, pp. S917–S922.

Suh, J.C., Lee, S. and Yoon, E.S. (1997), "New strategy for automated hazard analysis of chemical plants. Part 2: Reasoning algorithm and case study", *Journal of Loss Prevention in the Process Industries*, Vol. 10 No. 2, pp. 127–134.

Sukor, A.S.A., Zakaria, A., Rahim, N.A., Kamarudin, L.M., Setchi, R. and Nishizaki, H. (2019),

"A hybrid approach of knowledge-driven and data-driven reasoning for activity recognition in smart homes", *Journal of Intelligent and Fuzzy Systems*, Vol. 36 No. 5, pp. 4177–4188.

Suokas, J., Heino, P. and Karvonen, I. (1990), "Expert systems in safety management", *Journal of Occupational Accidents*, Vol. 12 No. 1–3, pp. 63–78.

Swann, C.D. and Preston, M.L. (1995), "Twenty-five years of HAZOPs", *Journal of Loss Prevention in the Process Industries*, Vol. 8 No. 6, pp. 349–353.

Tartir, S., Arpinar, I.B., Moore, M., Sheth, A.P. and Aleman-meza, B. (2005), "OntoQA: Metric-based ontology quality analysis", *IEEE ICDM 2005 Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, No. June 2014, available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.2087.

Tartir, S., Arpinar, I.B. and Sheth, A. (2010), "Ontological Evaluation and Validation", No. September, available at:https://doi.org/10.1007/978-90-481-8847-5.

Taylor, J.R. (1982), *Automatic Fault Tree Construction with RIKKE*, Vol. 2, Roskilde.

Taylor, J.R. (2017), "Automated HAZOP revisited", *Process Safety and Environmental Protection*, Institution of Chemical Engineers, Vol. 111, pp. 635–651.

Trammell, S.R. and Davis, B.J. (2001), "Using a modified Hazop/FMEA methodology for assessing system risk", *Proceedings - 2nd International Workshop on Engineering Management for Applied Technology, EMAT 2001*, pp. 47–53.

Trujillo, A., Kessler, W.S. and Gaither, R. (2015), "Common Mistakes When Conducting a HAZOP and How to Avoid Them", *Chemical Engineering*, No. December, available at: https://www.chemengonline.com/common-mistakes-conducting-hazop-avoid/.

Tsarkov, D. and Horrocks, I. (2006), "FaCT++ Description Logic Reasoner: System Description", in Furbach, U. and Shankar, N. (Eds.), *Automated Reasoning. IJCAR 2006*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 292–297.

Turney, R.D. (1991), "The Application of Total Quality Management to Hazard Studies and Their Recording", *International Journal of Quality ... Reliability Management*, Vol. 8 No. 6, available at:https://doi.org/10.1108/EUM0000000001641.

Tzinis, I. (2012), "Technology Readiness Level", Brian Dunbar, available at: http://www.nasa.gov/directorates/heo/scan/engineering/technology/technology_readiness_level (accessed 25 August 2021).

UN. (2019), *GLOBALLY HARMONIZED SYSTEM OF CLASSIFICATION AND LABELLING OF CHEMICALS (GHS)*, Eighth rev., New York and Geneva, available at: http://www.unece.org/fileadmin/DAM/trans/danger/publi/ghs/ghs_rev08/ST-SG-AC10-30-Rev8e.pdf.

Vaidhyanathan, R. and Venkatasubramanian, V. (1995), "Digraph-based models for automated HAZOP analysis", *Reliability Engineering and System Safety*, Vol. 50 No. 1, pp. 33–49.

Vaidhyanathan, R., Venkatasubramanian, V. and Dyke, F.T. (1996), "HAZOPExpert: An expert system for automating HAZOP analysis", *Process Safety Progress*, Vol. 15 No. 2, pp. 80–88.

Venkatasubramanian, V., Rengaswamy, R. and Kavuri, S.N. (2003), "A review of process fault detection and diagnosis part II: Qualitative models and search strategies", *Computers and Chemical Engineering*, Vol. 27 No. 3, pp. 313–326.

Venkatasubramanian, V., Zhao, J. and Viswanathan, S. (2000), "Intelligent systems for HAZOP analysis of complex process plants", *Computers and Chemical Engineering*, Vol. 24 No. 9–10, pp. 2291–2302.

Vrandecic, D., Vrandecic, D., Völker, J., Völker, J., Haase, P., Haase, P., Tran, D.T., *et al.* (2006), "A Metamodel for Annotations of Ontology Elements in OWL DL", *Proceedings of the 2nd Workshop on Ontologies and Meta-Modeling*, No. 109, pp. 109–123.

Wakeman, S.J., Chung, P.W.H., Rushton, A.G., Lees, F.P., Larkin, F.D. and McCoy, S.A. (1997), "Computer aided hazard identification: Fault propagation and fault-consequence scenario

filtering", *Institution of Chemical Engineers Symposium Series*, No. 141, pp. 305–316.

Wang, H., Chen, B., He, X., Tong, Q. and Zhao, J. (2009), "SDG-based HAZOP analysis of operating mistakes for PVC process", *Process Safety and Environmental Protection*, Vol. 87 No. 1, pp. 40–46.

Wang, H.H., Noy, N., Rector, A., Musen, M., Redmond, T., Rubin, D., Tu, S., *et al.* (2006), *Frames and OWL Side by Side*, *Presentation Abstracts*, available at:https://doi.org/10.1080/j.1440-1614.2006.01852.x.

Wiesner, A., Morbach, J., Yang, A., Bayer, B. and Marquardt, W. (2008), "OntoCAPE 2.0-Chemical Process Systems", Vol. 49 No. July, p. 149.

Wilson, R.J. (1996), *Introduction to Graph Theory*, 4th ed., Longman, available at: https://books.google.de/books?id=tSolAQAAIAAJ.

Wu, C.G., Xu, X., Zhang, B.K. and Na, Y.L. (2013), "Domain ontology for scenario-based hazard evaluation", *Safety Science*, Elsevier Ltd, Vol. 60, pp. 21–34.

Wygant, R.M. (1989), "CLIPS — A powerful development and delivery expert system tool", *Computers & Industrial Engineering*, Vol. 17 No. 1–4, pp. 546–549.

Yang, F., Shah, S.L. and Xiao, D. (2010), *SDG (Signed Directed Graph) Based Process Description and Fault Propagation Analysis for a Tailings Pumping Process*, *IFAC Proceedings Volumes (IFAC-PapersOnline)*, Vol. 13, IFAC, available at:https://doi.org/10.3182/20100802-3-ZA-2014.00011.

Zhai, Z., Ortega, J.F.M., Martínez, N.L. and Castillejo, P. (2018), "A rule-based reasoner for underwater robots using OWL and SWRL", *Sensors*, Vol. 18 No. 10, available at:https://doi.org/10.3390/s18103481.

Zhao, C., Bhushan, M. and Venkatasubramanian, V. (2005a), "Phasuite: An automated HAZOP analysis tool for chemical processes Part II: Implementation and case study", *Process Safety and Environmental Protection*, Vol. 83 No. 6 B, pp. 533–548.

Zhao, C., Bhushan, M. and Venkatasubramanian, V. (2005b), "Phasuite: An automated HAZOP analysis tool for chemical processes: Part I: Knowledge engineering framework", *Process Safety and Environmental Protection*, Vol. 83 No. 6 B, pp. 509–532.

Zhao, J., Cui, L., Zhao, L., Qiu, T. and Chen, B. (2009), "Learning HAZOP expert system by case-based reasoning and ontology", *Computers and Chemical Engineering*, Vol. 33 No. 1, pp. 371–378.

Appendix A Equipment entities

```
# === Describes the way the edge is connected
class MyPort:
    def __init__(self, name, port_instrumentation, connection_type):
        self.name = name
        self.port_instrumentation = port_instrumentation
        self.connection_type = connection_type
        self.onto_object = equipment_onto.Port(hasConnectionType=self.connection_type,
                                               hasName=self.name,
                                               portEquippedWithInstrumentation=self.port_instrumentation)
```

Appendix Listing A-1: Python class definition for creating the Port *Object*

```
class MyEquipmentEntity:
    def __init__(self,
                 equipment,
                 identifier,
                 boundary condition,
                 transportable,
                 operating_modes,
                 max_operating_pressure_in_barg,
                 max operating temperature in kelvin):
        self.name = equipment
        self.identifier = identifier
        self.max operating pressure in barg = max operating pressure in barg
        self.max operating temperature in kelvin = max operating temperature in kelvin
        self.onto_object = None
        self.instrumentation = []
        self.fixture = []
        self.piping = []
        self.subunit = []
        self.boundary condition = boundary condition
        self.material transfer equipment = []
        self.apparatus = []
        self.intended function = []
        self.transportable = transportable
        self.connected_plant_item = []
        self.control_instance = []
        self.operating_mode = operating_modes
        self.volume_of_enclosure = None
        self.volume_flow_of_transfer_equipment = None
        self.pressure volume product = None

    def set volume of enclosure(self, volume of enclosure):
        self.volume_of_enclosure = volume_of_enclosure

    def set_volume_flow_of_transfer_equipment(self, volume_flow):
        self.volume_flow_of_transfer_equipment = volume_flow

    def add_connected_plant_item(self, connected_plant_item):
        if connected plant item:
            self.connected plant item.append(connected plant item(None))

    def set control instance(self, control instance):
        if control_instance:
            self.control_instance.append(control_instance)

    def set_intended_function(self, intended_function):
        if intended_function:
            self.intended function.append(intended function(None))

    def set apparatus(self, apparatus):
        if apparatus:
            self.apparatus.append(apparatus(None))

    def add_fixture(self, fixture):
        if fixture:
            self.fixture.append(fixture(None))

    def add piping(self, piping):
        if piping:
            self.piping.append(piping(None))

    def add_instrumentation(self, instrumentation):
        if instrumentation:
            instrumentation instance = instrumentation[0](None)
            # Append unique tag of instrumentation as annotation to instrumentation instance
            if instrumentation[1]:
                instrumentation instance.comment = instrumentation[1]
            self.instrumentation.append(instrumentation instance)

    def add subunit(self, subunit):
        if subunit:
            self.subunit.append(subunit(None))

    def set material transfer equipment(self, material transfer equipment):
        if material transfer equipment:
            self.material transfer equipment.append(material transfer equipment(None))

    def determine pv product(self):
        self.pressure_volume_product = self.max_operating_pressure_in_barg * self.volume_of_enclosure

    def assemble_ontology_object(self, equipment_entity):
        self.onto_object = equipment_entity(hasFixture=self.fixture,
                                            hasInstrumentation=self.instrumentation,
                                            hasMaterialTransferEquipment=self.material transfer equipment,
                                            hasSubunit=self.subunit,
                                            hasApparatus=self.apparatus,
                                            hasPiping=self.piping,
                                            hasIntendedFunction=self.intended_function,
                                            hasConnectionToAdjacentPlantItem=self.connected_plant_item,
                                            hasMaximumOperatingPressureInBarGauge=self.max_operating_pressure_in_barg,
                                            hasMaximumOperatingTemperatureInKelvin=self.max_operating_temperature_in_kelvin,
                                            isTransportable=self.transportable,
                                            entityControlledBy=self.control_instance,
                                            hasPort=[],
                                            hasOperationMode=[])

    def set max operating pressure in barg(self, max operating pressure):
        self.max_operating_pressure_in_barg = max_operating_pressure
```

Appendix Listing A-2: Python class definition for creating the *Equipment Entity Object*

Appendix B Substances

```
class MySubstance:
    def __init__(self,
                 substance_name,
                 cas_number,
                 freezing_point,
                 boiling_point,
                 flash_point,
                 lower_explosion_limit,
                 upper_explosion_limit):

        self.name = substance_name
        self.cas_number = cas_number
        self.onto_object = None
        self.freezing_point = freezing_point
        self.boiling_point = boiling_point
        self.flash_point = flash_point
        self.lower_explosion_limit = lower_explosion_limit
        self.upper_explosion_limit = upper_explosion_limit
        self.task = []
        self.intended_state_of_aggregation = []
        self.hazard_class = []
        self.stability_reactivity_information = []

    def add_hazard_class(self, hazard_class):
        if hazard_class:
            for haz_c in hazard_class:
                self.hazard_class.append(haz_c(None))

    def add_stability_reactivity_information(self, stability_reactivity_information):
        if stability_reactivity_information:
            for sr_info in stability_reactivity_information:
                self.stability_reactivity_information.append(sr_info())

    def set_intended_state_of_aggregation(self, state_of_aggregation):
        if state_of_aggregation:
            self.intended_state_of_aggregation.append(state_of_aggregation())

    def set_substance_task(self, substance_task):
        self.task.append(substance_task())

    def assemble_onto_object(self, substance):
        self.onto_object = substance(hasSpecificTask=self.task,
                                     hasStabilityReactivityInformation=self.stability_reactivity_infor
mation,
                                     hasHazardClass=self.hazard_class,
                                     hasStateOfAggregation=self.intended_state_of_aggregation,
                                     hasBoilingPointInKelvin=self.boiling_point,
                                     hasFreezingPointInKelvin=self.freezing_point,
                                     hasFlashpointInKelvin=self.flash_point,
                                     hasUpperExplosionLimitInPercent=self.upper_explosion_limit,
                                     hasLowerExplosionLimitInPercent=self.lower_explosion_limit
                                     )
```

Appendix Listing B-1: Python class definition of the Substance Entity Object

Appendix C The main routine of the knowledge-based system

```
if   name   == '__main__':
    # === save all ontologies
    ontology operations.save ontology()
    ontology operations.save ontology as sql()
    upper_onto.determine_onto()

    start time = time.time()
    # === Process model
    process plant model = model.create hazid benchmark 1()

    deviation number = None
    stack elements = []

    # === Tearing strategy/strategy for defining order of process equipment/start-end point
    if len(process_plant_model.nodes) > 1:
        graph_type, newly_arranged_graphs, intersections = algorithm.determine_propagation_strategy(process_plant_model)
        print(list(newly arranged graphs))

    if config.EQUIPMENT BASED EVALUATION:
        for index in range(len(process plant model.nodes)):
            equipment_specific_prop_scenarios = []

            equipment_entity = process_plant_model.nodes[index]["data"]
            # Add port information to equipment entity
            for key, value in process_plant_model.nodes[index]["ports"].items():
                equipment_entity.onto_object.hasPort.append(value.onto_object)
                # Add port information to equipment entity
                if value.port instrumentation:
                    equipment entity.onto object.hasInstrumentation.append(value.port instrumentation)

            process_unit_obj = equipment_entity.onto_object.is_a[0]

            # === continue in case of source and sink, since they are not relevant in
            # equipment-based mode to save reasoner calls
            if process_unit_obj == equipment_onto.SinkEntity:
                continue

            substances = process plant model.nodes[index]["substances"]
            environment = process plant model.nodes[index]["environment"]
            deviations = config.deviation selector(process unit obj)

            # === Infer hazards
            infer.equipment_based_analysis(equipment_entity,
                                           deviations,
                                           substances,
                                           environment,
                                           equipment specific prop scenarios)

            stack elements.append({"{0}".format(equipment entity.name): equipment entity,
                                   pre_processing.DictName.scenario: equipment_specific_prop_scenarios})

    # === Entry point for hazard/malfunction propagation
    if config.PROPAGATION BASED EVALUATION:
        if graph type == algorithm.GraphType.SingleLineSystem:
            infer.propagation based analysis(process plant model, newly arranged graphs, stack elements)

        elif graph type == algorithm.GraphType.MultiCycleSystem:
            for index, cycle in enumerate(newly arranged graphs):
                # intersection of cycles
                intersection_node_index = cycle.index(intersections)

                # start first cycle one node after intersection and following nodes at intersection
                if index == 0:
                    intersection node index += 1

                # === Change order of graph [intersection + 1] as starting node
                new order = []
                for node in go.starting with(cycle, intersection node index):
                    new_order.append(node)

                infer.propagation_based_analysis(process_plant_model,
                                                 new order,
                                                 stack elements)

        elif graph type == algorithm.GraphType.JunctionSystem or \
                graph type == algorithm.GraphType.ComplexSystem or \
                graph_type == algorithm.GraphType.RecycleFlowSystem:
            for stream in newly_arranged_graphs:
                infer.propagation_based_analysis(process_plant_model, stream, stack_elements)
        else:
            print("Case not covered by any strategy!")

    # === remove duplicates and sort results table
    results = pre processing.cleanup result list(results)
    results = sorted(results, key=lambda k: k["process_equipment_id"])

    # === Hazop table
    if results:
        counter = 0
        for row in results:
            if row["cause"] or row["effect"] or row["consequence"]:
                output.hazop table.add row([counter,
                                            row["process_equipment"],
                                            row["process_equipment_id"],
                                            ', '.join(row["operation_mode"]),
                                            row["substance"],
                                            ', '.join(row["deviation"]),
                                            ', '.join(row["super_cause"]),
                                            ', '.join(row["cause"]),
                                            ', '.join(row["effect"]),
                                            ', '.join(row["consequence"]),
                                            ', '.join(row["safeguard"]),
                                            row["propagated"],
                                            ', '.join(row["risk"]),
                                            ])
                counter += 1

    print(output.hazop_table)

    output.create output()

    elapsed time = time.time() - start time
    print(elapsed time)

    ontology_operations.save_ontology()

    # ===Get errors
    print(list(default_world.inconsistent_classes()))
```

Appendix Listing C-1: Python code of the main routine of the knowledge-based system

Appendix D Equipment entity library

```
def source(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature):
    equipment = equipment onto.SourceEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature )
    equipment entity.set intended function(process onto.NoIntendedFunction)
    equipment_entity.set_apparatus(equipment_onto.NoApparatus)
    equipment_entity.set_control_instance(control_instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity

def sink(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature):
    equipment = equipment onto.SinkEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment_entity.set_intended_function(process_onto.NoIntendedFunction)
    equipment_entity.set_apparatus(equipment_onto.NoApparatus)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment entity

def connection pipe(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature):
    equipment = equipment onto.ConnectionPipeEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes, max_pressure, max_temperature)
    equipment_entity.set_intended_function(process_onto.Transport)
    equipment_entity.set_apparatus(equipment_onto.NoApparatus)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def tank truck(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature, volume of enclosure):
    equipment = equipment onto.TankTruckEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment_entity.set_volume_of_enclosure(volume_of_enclosure)
    equipment_entity.set_intended_function(process_onto.Loading)
    equipment_entity.set_apparatus(equipment_onto.PressureVessel)
    equipment_entity.add_piping(equipment_onto.TankTruckHose)
    equipment_entity.add_instrumentation((equipment_onto.ApiAdaptorValve, "{}-VA1".format(identifier)))
    equipment_entity.add_fixture(equipment_onto.Baffle)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment_entity

def storage_tank_1(identifier, circumstances, control_instance, transportable, operating_modes):
    equipment = equipment_onto.StorageTankEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes)
    equipment_entity.set_apparatus(equipment_onto.AtmosphericStorageTank)
    equipment entity.set intended function(process onto.Storage)
    equipment entity.add piping(equipment onto.BlanketingGasline)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add instrumentation((equipment onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def storage tank 2(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature, volume of enclosure):
    equipment = equipment onto.StorageTankEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment entity.set volume of enclosure(volume of enclosure)
    equipment entity.set apparatus(equipment onto.AtmosphericStorageTank)
    equipment entity.set intended function(process onto.Storage)
    equipment_entity.add_piping(equipment_onto.VentPipe)
    equipment_entity.add_instrumentation((equipment_onto.LevelIndicator, None))
    equipment entity.add instrumentation((equipment onto.HighLevelAlarm, None))
    # equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    # equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    # equipment entity.add instrumentation((equipment onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment entity.add subunit(equipment onto.ElectricalEnergySupply)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment_entity

def cooled_storage_tank_1(identifier, circumstances, control_instance, transportable, operating_modes):
    equipment = equipment onto.StorageTankEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add instrumentation((equipment onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment_entity.add_subunit(equipment_onto.CoolingSystem)
    equipment_entity.add_subunit(equipment_onto.ElectricalEnergySupply)
    equipment_entity.set_apparatus(equipment_onto.AtmosphericStorageTank)
    equipment_entity.add_piping(equipment_onto.BlanketingGasline)
    equipment_entity.add_piping(equipment_onto.TubeCoil)
    equipment entity.add instrumentation((equipment onto.Controller, None))
    equipment entity.set intended function(process onto.Storage)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment_entity

def cooled_storage_tank_2(identifier, circumstances, control_instance, transportable, operating_modes, max_pressure, max_temperature):
    equipment = equipment_onto.StorageTankEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes, max_pressure, max_temperature)
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add instrumentation((equipment onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment entity.add subunit(equipment onto.CoolingSystem)
    equipment entity.set apparatus(equipment onto.AtmosphericStorageTank)
    equipment_entity.add_piping(equipment_onto.BlanketingGasline)
    equipment_entity.add_fixture(equipment_onto.TubeCoil)
    equipment_entity.add_instrumentation((equipment_onto.LevelIndicatorController, None))
    equipment_entity.add_connected_plant_item(equipment_onto.FlowControlValve)
    equipment_entity.set_intended_function(process_onto.Storage)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity

def reactor(identifier, circumstances, control_instance, transportable, operating_modes):
    equipment = equipment_onto.ReactorEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes)
    equipment_entity.add_subunit(equipment_onto.ElectricalEnergySupply)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add instrumentation((equipment onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment entity.set apparatus(equipment onto.PressureVessel)
    equipment entity.set intended function(process onto.Reaction)
    equipment entity.set control instance(control instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity
```

Appendix Listing D-1: Equipment entity library I

```
def shell tube heat exchanger(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.ShellTubeHeatExchangerEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    # TODO: wieso nur 2 Anschlüsse
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.set_apparatus(equipment_onto.PressureVessel)
    equipment_entity.add_fixture(equipment_onto.Baffle)
    equipment entity.add piping(equipment onto.TubeBundle)
    equipment entity.set intended function(process onto.HeatTransfer)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity

def reboiler(identifier, circumstances, control_instance, transportable, operating_modes, max_pressure, max_temperature):
    equipment = equipment_onto.SteamDrivenReboilerEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment entity.set apparatus(equipment onto.PressureVessel)
    equipment entity.add fixture(equipment onto.Baffle)
    equipment entity.add piping(equipment onto.TubeBundle)
    equipment entity.add subunit(equipment onto.SteamSupply)
    equipment entity.set intended function(process onto.HeatTransfer)
    equipment entity.set control instance(control instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def settling tank(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature, volume of enclosure):
    equipment = equipment onto.SettlingTankEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    # equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    # equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    # equipment_entity.add_instrumentation((equipment_onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment_entity.add_piping(equipment_onto.BlanketingGasline)
    equipment_entity.add_instrumentation((equipment_onto.LevelIndicatorController, None))
    equipment_entity.add_instrumentation((equipment_onto.PressureIndicatorController, None))
    equipment entity.set apparatus(equipment onto.PressureVessel)
    equipment entity.set intended function(process onto.Separation)
    equipment entity.assemble ontology object(equipment)
    equipment entity.set control instance(control instance)
    return equipment_entity

def inertgas_blanketing(identifier, circumstances, control_instance, transportable, operating_modes, max_pressure, max_temperature):
    equipment = equipment_onto.InertgasBlanketingEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes, max_pressure, max_temperature)
    equipment_entity.add_instrumentation((equipment_onto.PressureControlValve, "{}-VA1".format(identifier)))
    equipment entity.add subunit(equipment onto.InertgasSupply)
    equipment entity.set apparatus(equipment onto.NoApparatus)
    equipment entity.add subunit(equipment onto.NonReturnValve)
    equipment entity.add piping(equipment onto.BlanketingGasline)
    equipment_entity.add_connected_plant_item(equipment_onto.PressureIndicatorController)
    equipment_entity.set_intended_function(process_onto.Inertization)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def pressure vessel(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.PressureVesselEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.set_apparatus(equipment_onto.PressureVessel)
    equipment_entity.set_intended_function(process_onto.Storage)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment entity

def pressure receiver(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature, volume):
    equipment = equipment onto.PressureReceiverEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes, max_pressure, max_temperature)
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment entity.set apparatus(equipment onto.PressureVessel)
    equipment entity.set intended function(process onto.Storage)
    equipment entity.set volume of enclosure(volume)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity

def plate_heat_exchanger(identifier, circumstances, control_instance, transportable, operating_modes):
    equipment = equipment onto.PlateHeatExchangerEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add subunit(equipment onto.SealingSystem)
    equipment entity.add fixture(equipment onto.PlatePackage)
    equipment entity.set apparatus(equipment onto.NoApparatus)
    equipment_entity.set_intended_function(process_onto.HeatTransfer)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment entity

def centrifugal pump 1(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.PumpEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.add_subunit(equipment_onto.ElectricalEnergySupply)
    equipment_entity.add_subunit(equipment_onto.SealingSystem)
    equipment_entity.set_material_transfer_equipment(equipment_onto.CentrifugalPump)
    equipment entity.add instrumentation((equipment onto.ElectricMotor, None))
    equipment entity.add instrumentation((equipment onto.Controller, None))
    equipment entity.set apparatus(equipment onto.PumpCasing)
    equipment entity.set intended function(process onto.MaterialTransfer)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity
```

Appendix Listing D-2: Equipment entity library II

```
def centrifugal pump 2(identifier,
                       circumstances,
                       control instance,
                       transportable,
                       operating_modes,
                       max_pressure,
                       max_temperature,
                       volume_flow):
    equipment = equipment onto.PumpEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment entity.set volume flow of transfer equipment(volume flow)
    equipment entity.set apparatus(equipment onto.PumpCasing)
    equipment entity.set intended function(process onto.MaterialTransfer)
    equipment_entity.set_material_transfer_equipment(equipment_onto.CentrifugalPump)
    equipment_entity.add_instrumentation((equipment_onto.ElectricMotor, None))
    equipment_entity.add_instrumentation((equipment_onto.Controller, None))
    equipment entity.add fixture(equipment onto.Impeller)
    # equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    # equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    # equipment entity.add instrumentation((equipment onto.BottomDrainValve, "{}-VA3".format(identifier)))
    equipment entity.add subunit(equipment onto.ElectricalEnergySupply)
    equipment entity.add subunit(equipment onto.SealingSystem)
    equipment entity.add subunit(equipment onto.NonReturnValve)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment entity

def screw compressor(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature, volume flow):
    equipment = equipment onto.CompressorEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.add_subunit(equipment_onto.ElectricalEnergySupply)
    equipment_entity.add_subunit(equipment_onto.SealingSystem)
    equipment_entity.add_subunit(equipment_onto.LubricationSystem)
    equipment entity.set material transfer equipment(equipment onto.ScrewCompressor)
    equipment entity.add instrumentation((equipment onto.ElectricMotor, None))
    equipment entity.add instrumentation((equipment onto.SpeedController, None))
    equipment entity.set apparatus(equipment onto.CompressorCasing)
    equipment_entity.set_intended_function(process_onto.MaterialTransfer)
    equipment_entity.set_volume_flow_of_transfer_equipment(volume_flow)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def piston compressor(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.CompressorEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.add_subunit(equipment_onto.ElectricalEnergySupply)
    equipment_entity.add_subunit(equipment_onto.SealingSystem)
    equipment_entity.add_subunit(equipment_onto.LubricationSystem)
    equipment_entity.set_material_transfer_equipment(equipment_onto.PistonCompressor)
    equipment_entity.add_instrumentation((equipment_onto.ElectricMotor, None))
    equipment entity.add instrumentation((equipment onto.SpeedController, None))
    equipment entity.set apparatus(equipment onto.CompressorCasing)
    equipment entity.set intended function(process onto.MaterialTransfer)
    equipment entity.set control instance(control instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def pneumatically_flow_control_valve_1(identifier, circumstances, control_instance, transportable, operating_modes, max_pressure, max_temperature):
    equipment = equipment_onto.ValveEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment entity.set apparatus(equipment onto.Body)
    equipment entity.set intended function(process onto.FluidControl)
    equipment entity.add instrumentation((equipment onto.PneumaticActuator, None))
    equipment_entity.add_instrumentation((equipment_onto.Controller, None))
    equipment_entity.add_subunit(equipment_onto.CompressedAirSupply)
    equipment_entity.add_subunit(equipment_onto.Bypass)
    equipment_entity.add_instrumentation((equipment_onto.FlowControlValve, "{}-VA1".format(identifier)))
    equipment entity.add connected plant item(equipment onto.LevelIndicatorController)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity

def pneumatically pressure control valve 1(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment_onto.ValveEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes)
    equipment entity.set apparatus(equipment onto.Body)
    equipment entity.set intended function(process onto.PressureControl)
    equipment entity.add instrumentation((equipment onto.PneumaticActuator, None))
    equipment entity.add instrumentation((equipment onto.Controller, None))
    equipment entity.add subunit(equipment onto.CompressedAirSupply)
    equipment entity.add instrumentation((equipment onto.PressureControlValve, "{}-VA1".format(identifier)))
    equipment entity.add connected plant item(equipment onto.PressureIndicatorController)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def manual three way valve(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.ValveEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.add instrumentation((equipment onto.ThreeWayValve, "{}-VA1".format(identifier)))
    equipment_entity.set_apparatus(equipment_onto.Body)
    equipment_entity.set_intended_function(process_onto.FluidControl)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity

def throttling valve(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature):
    equipment = equipment onto.ValveEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment_entity.add_instrumentation((equipment_onto.ThrottlingValve, "{}-VA1".format(identifier)))
    equipment_entity.set_apparatus(equipment_onto.Body)
    equipment_entity.set_intended_function(process_onto.FluidControl)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity
```

Appendix Listing D-3: Equipment entity library III

```
def manual valve(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.ValveEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment_entity.add_instrumentation((equipment_onto.ShutOffValve, "{}-VA1".format(identifier)))
    equipment_entity.set_apparatus(equipment_onto.Body)
    equipment_entity.set_intended_function(process_onto.FluidControl)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment entity


def air cooled condenser(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature):
    equipment = equipment onto.AirCooledCondenserEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes, max_pressure, max_temperature)
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add subunit(equipment onto.ElectricalEnergySupply)
    equipment entity.set apparatus(equipment onto.Casing)
    equipment entity.add fixture(equipment onto.FinnedCoil)
    equipment entity.set material transfer equipment(equipment onto.Fan)
    equipment entity.add instrumentation((equipment onto.ElectricMotor, None))
    equipment entity.add instrumentation((equipment onto.SpeedController, None))
    equipment entity.set intended function(process onto.Condensation)
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment entity


def shell tube evaporator(identifier, circumstances, control instance, transportable, operating modes, max pressure, max temperature, volume):
    equipment = equipment onto.ShellTubeEvaporatorEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment_entity.add_instrumentation((equipment_onto.InletValve, "{}-VA1".format(identifier)))
    equipment_entity.add_instrumentation((equipment_onto.OutletValve, "{}-VA2".format(identifier)))
    equipment_entity.set_apparatus(equipment_onto.PressureVessel)
    equipment_entity.add_fixture(equipment_onto.Baffle)
    equipment entity.add fixture(equipment onto.TubeBundle)
    equipment entity.set volume of enclosure(volume)
    equipment entity.set intended function(process onto.Evaporation)
    equipment entity.set control instance(control instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity


def fin_tube_evaporator(identifier, circumstances, control_instance, transportable, operating_modes):
    equipment = equipment_onto.FinTubeEvaporatorEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.add instrumentation((equipment onto.InletValve, "{}-VA1".format(identifier)))
    equipment entity.add instrumentation((equipment onto.OutletValve, "{}-VA2".format(identifier)))
    equipment entity.add subunit(equipment onto.ElectricalEnergySupply)
    equipment_entity.set_apparatus(equipment_onto.Casing)
    equipment_entity.add_fixture(equipment_onto.FinnedCoil)
    equipment_entity.set_material_transfer_equipment(equipment_onto.Fan)
    equipment_entity.add_instrumentation((equipment_onto.ElectricMotor, None))
    equipment_entity.add_instrumentation((equipment_onto.SpeedController, None))
    equipment_entity.set_intended_function(process_onto.Evaporation)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity


def wet_scrubber(identifier, circumstances, control_instance, transportable, operating_modes):
    equipment = equipment_onto.WetScrubberEntity
    equipment_entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating_modes)
    equipment_entity.set_apparatus(equipment_onto.PressureVessel)
    equipment entity.add fixture(equipment onto.LiquidDistributor)
    equipment entity.add fixture(equipment onto.PackedBed)
    equipment entity.set intended function(process onto.Purification)
    equipment entity.set control instance(control instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity


def cooling tower(identifier, circumstances, control instance, transportable, operating modes):
    equipment = equipment onto.CoolingTowerEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes)
    equipment entity.set apparatus(equipment onto.OpenVessel)
    equipment entity.add subunit(equipment onto.ElectricalEnergySupply)
    equipment entity.add fixture(equipment onto.LiquidDistributor)
    equipment_entity.add_fixture(equipment_onto.PackedBed)
    equipment_entity.add_fixture(equipment_onto.Basin)
    equipment entity.add instrumentation((equipment onto.ElectricMotor, None))
    equipment entity.add instrumentation((equipment onto.SpeedController, None))
    equipment entity.set intended function(process onto.HeatTransfer)
    equipment entity.set control instance(control instance)
    equipment entity.assemble ontology object(equipment)
    return equipment entity


def crude_stabilizer_column(identifier, circumstances, control_instance, transportable, operating_modes, max_pressure, max_temperature):
    equipment = equipment_onto.StabilizerColumnEntity
    equipment entity = MyEquipmentEntity(equipment, identifier, circumstances, transportable, operating modes, max pressure, max temperature)
    equipment entity.set apparatus(equipment onto.PressureVessel)
    equipment entity.set intended function(process onto.Stabilization)
    equipment entity.add fixture(equipment onto.Tray)
    equipment entity.add fixture(equipment onto.ChimneyTray)
    equipment_entity.add_instrumentation((equipment_onto.LevelIndicatorController, None))
    equipment_entity.set_control_instance(control_instance)
    equipment_entity.assemble_ontology_object(equipment)
    return equipment_entity
```

Appendix Listing D-4: Equipment entity library IV

Appendix E Case base for subsequent deviations

```
phase weight = 0.15
apparatus weight = 0.3
event weight = 0.4
equipment weight = 0.05
intended_function_weight = 0.1


class CaseAttributes(Enum):
    No = 1
    EquipmentEntity = 2
    Event = 3
    Apparatus = 4
    IntendedFunction = 5
    SubstancePhase = 6
    InferredDeviation = 7


propagation case base = [
    # === Pump
    {CaseAttributes.No: 1,
     CaseAttributes.EquipmentEntity: (equipment_onto.PumpEntity, equipment_weight),
     CaseAttributes.Event: (effect_onto.PumpRunningDry, event_weight),
     CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature,
                        deviation onto.HighVibration]},

    {CaseAttributes.No: 2,
     CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
     CaseAttributes.Event: (causes_onto.EntrainedAir, event_weight),
     CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.NoFlow,
                        deviation_onto.LowFlow]},

    {CaseAttributes.No: 3,
     CaseAttributes.EquipmentEntity: (equipment onto.StorageTankEntity, equipment weight),
     CaseAttributes.Event: (effect onto.VolumetricDisplacement, event weight),
     CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

    {CaseAttributes.No: 4,
     CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
     CaseAttributes.Event: (effect onto.FluidCirculatesInsidePump, event weight),
     CaseAttributes.Apparatus: (equipment onto.PumpCasing, apparatus weight),
     CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature]},

    {CaseAttributes.No: 5,
     CaseAttributes.EquipmentEntity: (None, equipment weight),
     CaseAttributes.Event: (effect onto.Overfilling, event weight),
     CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
     CaseAttributes.IntendedFunction: (None, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

    {CaseAttributes.No: 6,
     CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
     CaseAttributes.Event: (effect onto.Cavitation, event weight),
     CaseAttributes.Apparatus: (equipment onto.PumpCasing, apparatus weight),
     CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
     CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

    {CaseAttributes.No: 7,
     CaseAttributes.EquipmentEntity: (equipment_onto.PumpEntity, equipment_weight),
     CaseAttributes.Event: (causes onto.DeadHeadingOfPump, event weight),
     CaseAttributes.Apparatus: (equipment onto.PumpCasing, apparatus weight),
     CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
     CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

    {CaseAttributes.No: 8,
     CaseAttributes.EquipmentEntity: (equipment_onto.PumpEntity, equipment_weight),
     CaseAttributes.Event: (deviation_onto.NoFlow, event_weight),
     CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
     CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
     CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

    {CaseAttributes.No: 9,
     CaseAttributes.EquipmentEntity: (equipment_onto.PumpEntity, equipment_weight),
     CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
     CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

    {CaseAttributes.No: 10,
     CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
     CaseAttributes.Event: (deviation_onto.HighFlow, event_weight),
     CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighFlow]},

    {CaseAttributes.No: 11,
     CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
     CaseAttributes.Event: (deviation onto.OtherThanComposition, event weight),
     CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},

    {CaseAttributes.No: 12,
     CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
     CaseAttributes.Event: (deviation onto.HighTemperature, event weight),
     CaseAttributes.Apparatus: (equipment onto.PumpCasing, apparatus weight),
     CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
     CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature]},
```

Appendix Listing E-1: Case-base for identifying subsequent deviations I

```
{CaseAttributes.No: 13,
 CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighPressure, event weight),
 CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 14,
 CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.PumpCasing, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},

{CaseAttributes.No: 15,
 CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PumpCasing, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow, deviation onto.LowPressure]},

{CaseAttributes.No: 16,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.UnintendedExothermicPolymerization, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature, deviation onto.HighPressure,
                    deviation_onto.NoFlow]},

{CaseAttributes.No: 17,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect onto.UnintendedExothermicPolymerization, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature, deviation_onto.HighPressure,
                    deviation_onto.NoFlow]},

{CaseAttributes.No: 18,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.UnintendedExothermicPolymerization, event_weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature, deviation onto.HighPressure,
                    deviation_onto.NoFlow]},

{CaseAttributes.No: 19,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.UnintendedExothermicPolymerization, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature, deviation onto.HighPressure,
                    deviation onto.NoFlow]},

# === Overfilling
{CaseAttributes.No: 20,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.Overfilling, event_weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.ElsewhereFlow]},

{CaseAttributes.No: 21,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.Overfilling, event_weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.ElsewhereFlow]},

# === ConnectionPipeEntity
{CaseAttributes.No: 22,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (effect onto.UnintendedExothermicPolymerization, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 23,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (effect_onto.AbnormalEvaporation, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 24,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (effect_onto.AbnormalEvaporation, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 25,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (effect_onto.Fracture, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 26,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (effect onto.Fracture, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 27,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (effect onto.Fracture, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},
```

Appendix Listing E-2: Case-base for identifying subsequent deviations II

```
{CaseAttributes.No: 28,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (effect onto.Fracture, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 29,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (causes onto.WaterHammer, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighVibration]},

{CaseAttributes.No: 30,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (effect onto.BrittleFracture, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 31,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (effect onto.FatigueFracture, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 32,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (effect_onto.Fracture, event_weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 33,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (effect_onto.Fracture, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 34,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (effect_onto.Fracture, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 35,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 36,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 37,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 38,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighVibration, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

{CaseAttributes.No: 39,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation onto.HighVibration, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

{CaseAttributes.No: 40,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighVibration, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

{CaseAttributes.No: 41,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighFlow, deviation_onto.HighVibration]},

{CaseAttributes.No: 42,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation_onto.HighFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighFlow]},
```

Appendix Listing E-3: Case-base for identifying subsequent deviations III

```
{CaseAttributes.No: 43,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighFlow, deviation_onto.HighVibration]},

{CaseAttributes.No: 44,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow]},

{CaseAttributes.No: 45,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 46,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 47,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowPressure, event_weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 48,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowPressure, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 49,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowPressure, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 50,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.OtherThanComposition, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},

{CaseAttributes.No: 51,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.OtherThanComposition, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},

{CaseAttributes.No: 52,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.OtherThanComposition, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},

{CaseAttributes.No: 53,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.ReverseFlow]},

{CaseAttributes.No: 54,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation onto.LowPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.ReverseFlow]},

{CaseAttributes.No: 55,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature]},

{CaseAttributes.No: 56,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature]},

{CaseAttributes.No: 57,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature]},

{CaseAttributes.No: 58,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},
```

Appendix Listing E-4: Case-base for identifying subsequent deviations IV

```
{CaseAttributes.No: 59,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},

{CaseAttributes.No: 60,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},

{CaseAttributes.No: 61,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 62,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 63,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighPressure, event_weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Transport, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 64,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.ElsewhereFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 65,
 CaseAttributes.EquipmentEntity: (equipment_onto.ConnectionPipeEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.ElsewhereFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 66,
 CaseAttributes.EquipmentEntity: (equipment onto.ConnectionPipeEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure, deviation_onto.LowFlow]},

# === Pressure vessel
{CaseAttributes.No: 67,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.IncompleteEvaporation, event weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Evaporation, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},

{CaseAttributes.No: 68,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.AbnormalEvaporation, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 69,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.AbnormalEvaporation, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 70,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.HeatBuildUp, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 71,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.HeatBuildUp, event_weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 72,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.AbnormalEvaporation, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

# {CaseAttributes.No: 73,
#  CaseAttributes.EquipmentEntity: (None, equipment_weight),
#  CaseAttributes.Event: (effect_onto.PressureSurge, event_weight),
#  CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
#  CaseAttributes.IntendedFunction: (None, intended_function_weight),
#  CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
#  CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 74,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect_onto.AbnormalEvaporation, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},
```

Appendix Listing E-5: Case-base for identifying subsequent deviations V

```
# === Vacuum formation
{CaseAttributes.No: 75,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation_onto.LowTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 76,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

# === Internal Leakage
{CaseAttributes.No: 77,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (causes onto.InternalLeakage, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

# == Fracture -> Low level, nicht relevant, since leakage is relevant in case of a Fracture
{CaseAttributes.No: 78,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 79,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

# == Low pressure in Tank
{CaseAttributes.No: 80,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 81,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowPressure, event_weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 82,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowLevel, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 83,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowLevel, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 84,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation_onto.LowLevel, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},
{CaseAttributes.No: 85,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowLevel, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 86,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowLevel, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},
{CaseAttributes.No: 87,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation onto.LowLevel, event weight),
 CaseAttributes.Apparatus: (None, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 88,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 89,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 90,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.NoFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},
```

Appendix Listing E-6: Case-base for identifying subsequent deviations VI

```
{CaseAttributes.No: 91,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 92,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 93,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 94,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 95,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.NoFlow, event_weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 96,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.NoFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 97,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 98,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 99,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 100,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 101,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.OtherThanComposition, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.OtherThanComposition]},

{CaseAttributes.No: 102,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation onto.OtherThanComposition, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.OtherThanComposition]},

# === Compressor
{CaseAttributes.No: 103,
 CaseAttributes.EquipmentEntity: (equipment_onto.CompressorEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.NoFlow, event_weight),
 CaseAttributes.Apparatus: (equipment onto.CompressorCasing, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow, deviation onto.LowPressure]},

{CaseAttributes.No: 104,
 CaseAttributes.EquipmentEntity: (equipment_onto.CompressorEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.CompressorCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowTemperature]},

{CaseAttributes.No: 105,
 CaseAttributes.EquipmentEntity: (equipment_onto.CompressorEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.CompressorCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature]},

{CaseAttributes.No: 106,
 CaseAttributes.EquipmentEntity: (equipment onto.CompressorEntity, equipment weight),
 CaseAttributes.Event: (deviation_onto.LowTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.CompressorCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},] },
```

Appendix Listing E-7: Case-base for identifying subsequent deviations VII

```
{CaseAttributes.No: 107,
 CaseAttributes.EquipmentEntity: (equipment onto.CompressorEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowPressure, event weight),
 CaseAttributes.Apparatus: (equipment_onto.CompressorCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow, deviation_onto.LowPressure]},

{CaseAttributes.No: 108,
 CaseAttributes.EquipmentEntity: (equipment onto.CompressorEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.CompressorCasing, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow, deviation_onto.LowPressure]},

{CaseAttributes.No: 109,
 CaseAttributes.EquipmentEntity: (equipment onto.CompressorEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.CompressorCasing, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.MaterialTransfer, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

# === Valve
{CaseAttributes.No: 110,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.Cavitation, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighVibration]},

# {CaseAttributes.No: 111,
#  CaseAttributes.EquipmentEntity: (None, equipment_weight),
#  CaseAttributes.Event: (effect onto.PressureSurge, event weight),
#  CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
#  CaseAttributes.IntendedFunction: (None, intended function weight),
#  CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
#  CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 112,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (effect_onto.Cavitation, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

{CaseAttributes.No: 113,
 CaseAttributes.EquipmentEntity: (equipment_onto.AirCooledCondenserEntity, equipment_weight),
 CaseAttributes.Event: (causes_onto.NonCondensables, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Casing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Condensation, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 114,
 CaseAttributes.EquipmentEntity: (equipment onto.PumpEntity, equipment weight),
 CaseAttributes.Event: (effect_onto.UnintendedExothermicPolymerization, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PumpCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Transport, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

# === Compressor
{CaseAttributes.No: 115,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect_onto.ExcessiveDischargeTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.CompressorCasing, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature]},

{CaseAttributes.No: 116,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.IncreasedOilDischarge, event weight),
 CaseAttributes.Apparatus: (equipment onto.CompressorCasing, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.MaterialTransfer, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature]},

# === Dev to Dev Cases=========================================================
{CaseAttributes.No: 117,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},

{CaseAttributes.No: 118,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},

{CaseAttributes.No: 119,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowTemperature]},

{CaseAttributes.No: 120,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighTemperature]},

{CaseAttributes.No: 121,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature]},
```

Appendix Listing E-8: Case-base for identifying subsequent deviations VIII

```
{CaseAttributes.No: 122,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighTemperature, event weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature]},

{CaseAttributes.No: 123,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.NoFlow]},

{CaseAttributes.No: 124,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 125,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 126,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighPressure, event_weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 127,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighPressure, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 128,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighPressure, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 129,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighFlow]},

{CaseAttributes.No: 130,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighFlow]},

{CaseAttributes.No: 131,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighFlow]},

{CaseAttributes.No: 132,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 133,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation onto.LowPressure, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 134,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowPressure, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 135,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 136,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow]},
```

Appendix Listing E-9: Case-base for identifying subsequent deviations IX

```
{CaseAttributes.No: 137,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow]},

{CaseAttributes.No: 138,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighVibration, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighVibration]},

{CaseAttributes.No: 139,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighVibration, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

{CaseAttributes.No: 140,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighVibration, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighVibration]},

{CaseAttributes.No: 141,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.OtherThanComposition, event_weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.OtherThanComposition]},

{CaseAttributes.No: 142,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.OtherThanComposition, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.OtherThanComposition]},

{CaseAttributes.No: 143,
 CaseAttributes.EquipmentEntity: (equipment_onto.ValveEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.OtherThanComposition, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},

{CaseAttributes.No: 144,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.Body, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowPressure]},

{CaseAttributes.No: 145,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow]},

{CaseAttributes.No: 146,
 CaseAttributes.EquipmentEntity: (equipment onto.ValveEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.Body, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowFlow, deviation_onto.LowPressure]},

{CaseAttributes.No: 147,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 148,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation onto.ElsewhereFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 149,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.ElsewhereFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 150,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.ElsewhereFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowLevel]},

{CaseAttributes.No: 151,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation_onto.ElsewhereFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 152,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},
```

Appendix Listing E-10: Case-base for identifying subsequent deviations X

```
{CaseAttributes.No: 153,
 CaseAttributes.EquipmentEntity: (equipment onto.StorageTankEntity, equipment weight),
 CaseAttributes.Event: (effect onto.AbnormalEvaporation, event weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 154,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 155,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighTemperature, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 156,
 CaseAttributes.EquipmentEntity: (equipment onto.InertgasBlanketingEntity, equipment weight),
 CaseAttributes.Event: (causes onto.IncreasedInletPressure, event weight),
 CaseAttributes.IntendedFunction: (process onto.Inertization, intended function weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 157,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 158,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 159,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighTemperature, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 160,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighTemperature, event weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 161,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighLevel]},

{CaseAttributes.No: 162,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighLevel]},

{CaseAttributes.No: 163,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighLevel]},

{CaseAttributes.No: 164,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 165,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 166,
 CaseAttributes.EquipmentEntity: (None, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighLevel, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 167,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation_onto.HighLevel, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 168,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation_onto.HighLevel, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},
```

Appendix Listing E-11: Case-base for identifying subsequent deviations XI

```
{CaseAttributes.No: 169,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighLevel, event weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 170,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighLevel, event weight),
 CaseAttributes.Apparatus: (equipment onto.AtmosphericStorageTank, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Storage, intended function weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

{CaseAttributes.No: 171,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (deviation onto.HighLevel, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},
# === Source
{CaseAttributes.No: 172,
 CaseAttributes.EquipmentEntity: (equipment onto.SourceEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (None, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.NoFlow]},

{CaseAttributes.No: 173,
 CaseAttributes.EquipmentEntity: (equipment_onto.SourceEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.LowFlow, event_weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.IntendedFunction: (None, intended function weight),
 CaseAttributes.SubstancePhase: (None, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowFlow]},

{CaseAttributes.No: 174,
 CaseAttributes.EquipmentEntity: (equipment_onto.SourceEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (None, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighFlow]},

{CaseAttributes.No: 175,
 CaseAttributes.EquipmentEntity: (equipment_onto.SourceEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.OtherThanComposition, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (None, intended_function_weight),
 CaseAttributes.SubstancePhase: (None, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},
# === Settling tank
{CaseAttributes.No: 176,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect onto.BacteriaGrowth, event weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighCorrosion]},

{CaseAttributes.No: 177,
 CaseAttributes.EquipmentEntity: (equipment onto.SettlingTankEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.LowFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (process_onto.Separation, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Liquid, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 178,
 CaseAttributes.EquipmentEntity: (equipment onto.SettlingTankEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.NoFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Separation, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation_onto.LowLevel]},

{CaseAttributes.No: 179,
 CaseAttributes.EquipmentEntity: (equipment onto.SettlingTankEntity, equipment weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Separation, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Liquid, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighLevel, deviation onto.HighFlow]},

{CaseAttributes.No: 180,
 CaseAttributes.EquipmentEntity: (equipment_onto.SettlingTankEntity, equipment_weight),
 CaseAttributes.Event: (deviation onto.HighFlow, event weight),
 CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
 CaseAttributes.IntendedFunction: (process onto.Separation, intended function weight),
 CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighLevel, deviation onto.HighFlow]},
# === Inertgas blanketing unit
{CaseAttributes.No: 181,
 CaseAttributes.EquipmentEntity: (equipment_onto.InertgasBlanketingEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.NoFlow, event_weight),
 CaseAttributes.IntendedFunction: (process_onto.Inertization, intended_function_weight),
 CaseAttributes.Apparatus: (equipment onto.NoApparatus, apparatus weight),
 CaseAttributes.SubstancePhase: (substance onto.Gaseous, phase weight),
 CaseAttributes.InferredDeviation: [deviation onto.LowPressure]},

{CaseAttributes.No: 182,
 CaseAttributes.EquipmentEntity: (equipment_onto.InertgasBlanketingEntity, equipment_weight),
 CaseAttributes.Event: (deviation_onto.HighFlow, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.NoApparatus, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Inertization, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Gaseous, phase_weight),
 CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

{CaseAttributes.No: 183,
 CaseAttributes.EquipmentEntity: (None, equipment weight),
 CaseAttributes.Event: (effect_onto.BacteriaGrowth, event_weight),
 CaseAttributes.Apparatus: (equipment_onto.AtmosphericStorageTank, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Storage, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.HighCorrosion]},

{CaseAttributes.No: 184,
 CaseAttributes.EquipmentEntity: (equipment onto.SettlingTankEntity, equipment weight),
 CaseAttributes.Event: (causes onto.ExcessiveInflow, event weight),
 CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
 CaseAttributes.IntendedFunction: (process_onto.Separation, intended_function_weight),
 CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
 CaseAttributes.InferredDeviation: [deviation_onto.OtherThanComposition]},
```

Appendix Listing E-12: Case-base for identifying subsequent deviations XII

```
{CaseAttributes.No: 185,
     CaseAttributes.EquipmentEntity: (equipment onto.SteamDrivenReboilerEntity, equipment weight),
     CaseAttributes.Event: (effect onto.IncreasedHeatingCapacity, event weight),
     CaseAttributes.Apparatus: (equipment_onto.PressureVessel, apparatus_weight),
     CaseAttributes.IntendedFunction: (process_onto.Evaporation, intended_function_weight),
     CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighTemperature]},

    {CaseAttributes.No: 186,
     CaseAttributes.EquipmentEntity: (equipment onto.StabilizerColumnEntity, equipment weight),
     CaseAttributes.Event: (causes onto.DeliveryOfHighVolatilityComponents, event weight),
     CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
     CaseAttributes.IntendedFunction: (process onto.Stabilization, intended function weight),
     CaseAttributes.SubstancePhase: (substance_onto.Multiphase, phase_weight),
     CaseAttributes.InferredDeviation: [deviation_onto.HighPressure]},

    {CaseAttributes.No: 187,
     CaseAttributes.EquipmentEntity: (equipment onto.SteamDrivenReboilerEntity, equipment weight),
     CaseAttributes.Event: (effect onto.HeatBuildUp, event weight),
     CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
     CaseAttributes.IntendedFunction: (process onto.Evaporation, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
     CaseAttributes.InferredDeviation: [deviation onto.HighPressure]},

    {CaseAttributes.No: 188,
     CaseAttributes.EquipmentEntity: (None, equipment weight),
     CaseAttributes.Event: (causes onto.InternalLeakage, event weight),
     CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
     CaseAttributes.IntendedFunction: (None, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
     CaseAttributes.InferredDeviation: [deviation onto.OtherThanComposition]},

    {CaseAttributes.No: 189,
     CaseAttributes.EquipmentEntity: (equipment_onto.SteamDrivenReboilerEntity, equipment_weight),
     CaseAttributes.Event: (causes_onto.LessSteamFlow, event_weight),
     CaseAttributes.Apparatus: (equipment onto.PressureVessel, apparatus weight),
     CaseAttributes.IntendedFunction: (process onto.Evaporation, intended function weight),
     CaseAttributes.SubstancePhase: (substance onto.Multiphase, phase weight),
     CaseAttributes.InferredDeviation: [deviation onto.LowTemperature]},
```

Appendix Listing E-13: Case-base for identifying subsequent deviations XIII

```
def calculate similarity(new case, old case):
    if new case is None or old case is None:
        print("Similarity could not be calculated because of None")
        return 0
    else:
        attr_list = [CaseAttributes.EquipmentEntity,
                     CaseAttributes.Event,
                     CaseAttributes.Apparatus,
                     CaseAttributes.IntendedFunction,
                     CaseAttributes.SubstancePhase]

        similarities = []
        weights = []
        sum_of_weights = 0

        for attr in attr_list:
            if old case.get(attr)[0]:
                similarities.append(similar(new case.get(attr), old case.get(attr)[0]))
            else:
                similarities.append(0)

            weights.append(old_case.get(attr)[1])
            sum_of_weights += old_case.get(attr)[1]

        similarity_products = [a * b for a, b in zip(similarities, weights)]
        sum_ = sum(similarity_products)

        # Overall similarity
        similarity = 1 / sum of weights * sum
        return similarity


def match_case_with_cb(current_case, case_base):
    list_similarities = []

    for case in case_base:
        list similarities.append((calculate similarity(current case, case), case.get(CaseAttributes.InferredDeviation)))

    res list = [x[0] for x in list similarities]

    max_similarity_measure = max(res_list)

    if max_similarity_measure >= 0.75:
        index = res_list.index(max_similarity_measure)
        relevant deviation = list similarities[index][1]
    else:
        relevant deviation = None

    return relevant deviation

]
```

Appendix Listing E-14: Functions for calculating the similarity measure

Appendix F Example runtime Hermit reasoner

Machine A					Machine B				
standard relations		inverse relations		CBR	NO inverse		inverse relations		CBR
Amount	155	155		36	No.	155	155		108
ø	10,3876288	28,47511381		0,005156028	ø	5,8157513	10,29335418		0,001802185
2,651015	10,889943	3,499647	52,828018	0,006533	1,500469	6,519138	1,568635	12,386779	0,001001	0,002002
3,091001	11,784997	3,189999	31,436804	0,008083	1,519346	6,672483	1,675943	12,168896	0,001002	0,001
3,083103	10,926003	3,198	32,593871	0,002044	1,676491	6,56579	1,820953	12,277806	0,002001	0,001
4,512769	10,988996	3,511001	33,425206	0,01857	1,721507	6,909361	1,786718	12,367579	0,002002	0,001
3,907001	11,581001	3,609	35,669978	0,006999	1,67054	6,72236	1,914687	12,325543	0,001001	0,002002
4,333079	11,243002	3,131543	34,410288	0,005	1,690467	7,031641	1,990757	12,451874	0,001001	0,002002
5,059046	19,899175	3,082001	34,938855	0,005001	1,735205	7,233076	1,912755	12,39826	0,002002	0,002001
2,990997	14,692509	3,239999	33,884888	0,002996	1,718702	6,754437	1,945808	12,606731	0,002001	0,001002
3,047998	11,468983	3,898579	33,317036	0,001997	1,709924	7,188739	1,880714	12,643154	0,001001	0,002001
3,222999	12,96257	3,255999	35,788072	0,002998	1,795295	7,03138	1,897091	12,809017	0,002002	0,001001
3,414995	12,799659	3,253	38,583811	0,001	1,828835	6,921494	1,975464	13,675156	0,002002	0,002002
3,259988	14,490119	3,782003	34,787113	0,001998	1,796058	7,006649	2,635158	13,248773	0,001001	0,002003
3,232018	15,764791	4,091038	36,268796	0,001	1,874541	7,142155	2,149179	13,528257	0,002002	0,001001
3,613999	13,254179	3,809948	35,218653	0,008545	1,829734	7,248146	2,224361	13,231443	0,002002	0,002516
3,273004	14,351683	3,918219	34,905924	0,015609	1,858854	7,418525	2,180086	13,703735	0,002002	0,001
3,284993	12,775148	3,539871	38,0983	0,001999	1,835417	7,456657	2,14705	13,333557	0,001001	0,002013
4,086653	12,697525	3,504999	35,602025	0,003999	1,91217	7,53291	2,276502	14,283946	0,001	0,002021
3,562001	13,61087	4,025001	37,309592	0,001999	1,93684	7,42347	2,146387	14,12805	0,002011	0,002186
3,404	13,015	3,509056	39,650683	0,001998	1,995013	7,50087	2,180588	14,174839	0,001	0,002017
3,387001	12,422999	3,68104	35,834894	0,002002	1,984312	7,777798	2,383082	14,346457	0,001	0,001
3,608001	12,676	4,031002	41,866881	0,002	1,941862	8,10215	2,259879	14,442914	0,002002	0,002013
3,640001	12,813	3,916	36,757627	0,003	1,856416	8,056015	2,188477	14,505981	0,002003	0,00201
4,098001	12,473	3,857975	38,660322	0,002999	2,00893	7,81665	2,287214	14,329652	0,001	0,001
4,130527	12,815019	4,466002	37,057293	0,010075	2,136631	8,159313	2,526079	14,001022	0,002003	0,001
3,744018	12,48451	4,766001	35,875657	0,00809	2,137423	7,809317	2,535537	14,052629	0,002001	0,006341
3,82456	12,48298	5,144682	38,801273	0,002042	2,215526	7,783711	2,511975	14,109945	0,002002	0,002
4,060979	13,298999	4,825785	49,62907	0,006516	2,107141	7,93862	2,49471	14,661521	0,002002	0,001001
3,800329	12,945978	4,672999	60,849319	0,010125	2,210299	8,224288	2,664616	14,908866	0,001001	0,002002
4,54	12,966999	6,746	55,022697	0,008078	2,518804	8,056167	3,073652	14,163377	0,002001	0,002002
4,400001	12,916979	5,9169	57,991078	0,002059	2,64485	7,818875	3,935615	14,503237	0,002002	0,001001
4,498	12,715023	5,679994	60,302955	0,008054	2,744092	7,549776	3,466072	14,364289	0,002002	
5,760266	13,97698	6,566281	58,226837	0,002043	2,60292	7,539534	3,668794	14,786371	0,002002	
5,132978	14,379998	9,127572	52,564106	0,008066	2,693906	7,773049	3,587792	14,484773	0,002002	
5,132981	14,058002	8,083764	55,834911	0,002041	3,048901	7,67433	3,885293	14,627992	0,002002	
4,846	13,20202	6,770792	55,468193	0,008031	2,757304	7,675409	3,907304	16,073722	0,002002	
5,299996	13,680001	6,41622	52,705774	0,002028	2,853785	7,618894	4,077027	14,773425	0,002003	
5,23602	13,717999	7,39385	55,840811		2,972897	7,693147	4,140986	14,866389	0,002001	
5,469999	13,406021	7,837248	53,134195		2,841819	7,833444	4,249434	14,376156	0,001001	
4,95998	13,692985	7,334881	38,246178		3,104534	8,314514	4,451375	14,554519	0,002002	
5,318999	13,521166	7,831669	39,675927		3,020329	8,055108	4,402042	15,051734	0,002001	
5,25915	15,836869	7,458562	54,592969		3,007999	8,125608	4,377421	15,509188	0,001001	
5,107795	13,712	7,211583	58,543774		3,12352	7,999121	4,454597	14,897325	0,002003	
6,323529	13,60852	9,395949	56,297405		3,4516	8,517842	5,464034	14,910998	0,001	
6,123	13,316999	10,216139	48,425014		3,627247	8,114793	5,616037	15,08307	0,007016	
6,325062	14	10,549624	47,21894		3,798787	8,114895	7,323473	15,194893	0,001	
6,599977	13,967	10,356344	40,361938		3,921983	8,161115	5,927365	14,755474	0,001	
6,81398	14,419053	10,44673	37,936804		3,738242	7,937444	5,63026	14,659077	0,005005	
6,954995	13,528997	9,559068	41,207078		3,795519	8,213537	5,770043	14,915171	0,001001	
6,304114	13,383	9,673524	44,448406		3,735545	8,332846	6,351125	15,33825	0,002002	
7,121794	14,721999	9,365519	42,054829		3,794509	8,814168	5,831302	15,439568	0,001001	
6,608996	14,811579	9,56715	43,261668		3,741005	8,115321	5,833489	15,507744	0,002001	
6,584001	14,911022	11,06967	43,515856		3,989176	8,50001	6,585537	15,112001	0,002002	
7,26	15,201978	12,049998	39,707137		4,457733	8,517761	6,743309	15,759409	0,002002	
7,400608	15,062	11,85503	46,213168		4,304921	8,882881	6,710644	15,792441	0,002002	
7,478998	14,833519	13,929854	40,794846		4,313023	8,443891	6,848508	16,69644	0,001	
8,079	16,063001	12,535562	40,49235		4,529473	8,731424	8,107003	16,449739	0,002002	
8,185023	18,350038	12,573845	42,511578		4,758285	8,763494	7,890587	18,109574	0,001	
7,825976	15,558022	14,91062	44,038342		4,604976	8,877826	8,418181	16,511141	0,005005	
8,569001	16,531155	13,449132	41,885039		4,808189	8,853663	7,618595	16,794466	0,003354	
9,475274	16,508999	12,710696	44,230938		4,724841	8,947826	7,698925	17,029009	0,001002	
7,945999	15,792102	13,236578	43,337056		4,715347	8,886258	8,217066	17,263749	0,002001	
8,701619	17,519095	13,771136	46,763937		4,84127	9,287388	8,028289	17,405937	0,002001	
9,375998	17,713655	14,54305	46,831189		4,740063	9,246164	8,264059	17,23155	0,002001	
8,201196	16,349281	14,682221	45,016611		4,764279	9,433298	8,060257	17,950861	0,002003	
8,552018	16,35802	13,307602	46,218267		4,857529	9,936027	8,731927	17,518293	0,002002	
7,752977	16,803378	13,896553	45,502573		4,987319	9,384155	8,295023	17,686555	0,002002	
8,774822	16,572859	14,560859	47,840198		4,993938	9,634181	8,484357	17,640695	0,002002	
9,112999	18,391926	19,916819	74,292145		5,233037	9,960713	8,937382	18,017566	0,002002	
9,570001	17,272391	46,607015	56,297556		5,47764	10,067513	9,455155	18,345644	0,001	
9,622005	18,014803	41,139207	52,1139		5,695146	9,283489	9,628569	18,414116	0,002002	
9,240997	20,174979	39,484173	49,252495		5,635852	10,31804	9,973281	18,535831	0,002002	
10,072981	16,721076	37,596767	51,190627		5,735937	9,615992	10,238559	18,461076	0,001002	
9,424536	18,184169	39,671934	56,740556		5,775364	9,703006	10,567912	19,035464	0,001001	
9,884979	19,073507	37,670285	55,414416		5,807711	9,665342	10,298002	19,521961	0,002001	
9,719022	18,549001	45,586878	55,860567		5,614778	10,225256	10,493692	19,383178	0,002002	
9,983079	21,073931	30,115517	53,267465		5,852828	10,357481	10,205259	19,316469	0,001001	
11,008977	18,010005	33,336426	56,162602		6,699501	10,047776	12,091615	19,395838	0,001	
	24,075946		54,615312			10,212316		19,975141	0,002002	

Appendix Listing F-1: Run time semantic reasoner and case-based reasoning

Appendix G Formal ontology class definitions of HazopCoreConcept

```
# ===Cause ======================================
class Cause(Thing):
    pass

Cause.label = ["cause"]
Cause.comment = ["Initiating event in the sequence of events of a scenario", "represents an event or situation"]

class UnderlyingCause(Thing):
    pass

UnderlyingCause.label = ["underlying cause"]
UnderlyingCause.comment = ["Underlying event in the sequence of events of a scenario",
                           "represents ambient conditions, systemic and organizational causes, "
                           "and causes arising from latent human errors"]

class causeInvolvesEquipmentEntity(Cause >> equipment_onto.EquipmentEntity):
    pass

class causeInvolvesSubstance(Cause >> substance onto.Substance):
    pass

class underlyingcauseInvolvesEquipmentEntity(UnderlyingCause >> equipment_onto.EquipmentEntity):
    pass

class underlyingcauseInvolvesSubstance(UnderlyingCause >> substance_onto.Substance):
    pass

class isCauseOfDeviation(Cause >> deviation onto.Deviation):
    pass

class causeInvolvesSecondDeviation(Cause >> deviation_onto.Deviation):
    pass

class isUnderlyingcauseOfCause(UnderlyingCause >> Cause):
    pass

class causeRequiresBoundaryCondition(Cause >> boundary onto.BoundaryCondition):
    pass

class causeInvolvesSiteInformation(Cause >> site information.AmbientInformation):
    pass

class underlyingcauseRequiresBoundaryCondition(UnderlyingCause >> boundary_onto.BoundaryCondition):
    pass

# === Effect ======================================
class Effect(Thing):
    pass

Effect.label = ["effect"]
Effect.comment = ["describe intermediate events that lie in the event sequence between"
                  "the fault event (deviation) and hazardous event (consequence)"]

class effectInvolvesEquipmentEntity(Effect >> equipment onto.EquipmentEntity):
    pass

class effectInvolvesSiteInformation(Effect >> site information.AmbientInformation):
    pass

class isEffectOfDeviation(Effect >> deviation_onto.Deviation):
    pass

class effectInvolvesSecondDeviation(Effect >> deviation onto.Deviation):
    pass

class effectInvolvesSubstance(Effect >> substance onto.Substance):
    pass

class effectRequiresBoundaryCondition(Effect >> boundary_onto.BoundaryCondition):
    pass

class effectImpliedByCause(Effect >> causes onto.Cause):
    pass

class effectOfPropagatedCause(Effect >> bool, FunctionalProperty):
    pass

class effectImpliedByUnderlyingcause(Effect >> causes_onto.UnderlyingCause):
    pass

# === Consequence ======================================
class Consequence(Thing):
    pass

Consequence.label = ["consequence"]
Consequence.comment = ["a consequence represents a hazardous event which is synonymous with a loss event"]

class consequenceInvolvesEquipmentEntity(Consequence >> equipment_onto.EquipmentEntity):
    pass

class consequenceInvolvesSubstance(Consequence >> substance_onto.Substance):
    pass

class isConsequenceOfDeviation(Consequence >> deviation onto.Deviation):
    pass

class isConsequenceOfEffect(Consequence >> effect_onto.Effect):
    pass

class consequenceImpliedByCause(Consequence >> causes_onto.Cause):
    pass

class isSubsequentConsequence(Consequence >> Consequence, AsymmetricProperty):
    pass

class consequenceRequiresBoundaryCondition(Consequence >> boundary_onto.BoundaryCondition):
    pass
```

Appendix Listing G-1: Class definitions and relationships of HazopCoreConcepts I

```
# === Safeguard ========================================
class Safeguard(Thing):
    pass

Safeguard.label = ["safeguard"]
Safeguard.comment = ["CCPS glossary: 'Any device, system, or action that interrupts the chain of events "
                     "following an initiating event or that mitigates the consequences.'"]

class safeguardOfDeviation(Safeguard >> deviation onto.Deviation):
    pass

class safeguardPreventsCause(Safeguard >> causes onto.Cause):
    pass

class safeguardPreventsUnderlyingCause(Safeguard >> causes_onto.UnderlyingCause):
    pass

class safeguardMitigatesConsequence(Safeguard >> consequence onto.Consequence):
    pass

class safeguardInvolvesSubstance(Safeguard >> substance onto.Substance):
    pass

class safeguardPreventsEffect(Safeguard >> effect_onto.Effect):
    pass

class safeguardInvolvesEquipmentEntity(Safeguard >> equipment onto.EquipmentEntity):
    pass

class safeguardInvolvesBoundaryCondition(Safeguard >> boundary onto.BoundaryCondition):
    pass

class safeguardDependsOnRiskCategory(Safeguard >> risk_assessment_onto.RiskCategory):
    pass

class impliesSafeguard(Safeguard >> Safeguard, AsymmetricProperty):
    pass

# === Disjoint statement
AllDisjoint([deviation_onto.Deviation,
             causes_onto.Cause,
             causes_onto.UnderlyingCause,
             effect_onto.Effect,
             consequence_onto.Consequence,
             Safeguard,
             risk assessment onto.Likelihood,
             risk assessment onto.SeverityCategory,
             risk assessment onto.RiskCategory])

# === Deviation ========================================
class Deviation(Thing):
    pass

Deviation.label = ["deviation"]
Deviation.comment = ["describes (process) deviation from intention",
                     "CCPS glossary: 'process condition outside of established design limits'",
                     "describes fault event in the sequence of events of a scenario"]

class hasGuideword(Deviation >> Guideword):
    pass

class hasParameter(Deviation >> Parameter):
    pass
```

Appendix Listing G-2: Class definitions and relationships of HazopCoreConcepts II

Appendix H Formal ontology class definitions deviations

```
# === Guideword ========================================
class Guideword(Thing):
    pass

class More(Guideword):
    pass

class Less(Guideword):
    pass

class No(Guideword):
    pass

class PartOf(Guideword):
    pass

class AsWellAs(Guideword):
    pass

class WhereElse(Guideword):
    pass

class OtherThan(Guideword):
    pass

class Reverse(Guideword):
    pass
# === Parameters =======================================
class Parameter(Thing):
    pass

class Pressure(Parameter):
    pass

class Temperature(Parameter):
    pass

class Level(Parameter):
    pass

class Flow(Parameter):
    pass

class Material(Parameter):
    pass

class Vibration(Parameter):
    pass

class Composition(Parameter):
    pass

class Corrosion(Parameter):
    pass

class Time(Parameter):
    pass
# === Deviation ========================================
class Deviation(Thing):
    pass
Deviation.label = ["deviation"]
Deviation.comment = ["describes (process) deviation from intention", "CCPS glossary: 'process condition outside of established design limits'",
                     "describes fault event in the sequence of events of a scenario"]

class hasGuideword(Deviation >> Guideword):
    pass

class hasParameter(Deviation >> Parameter):
    pass

class HighCorrosion(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(More) & hasParameter.some(Corrosion))]
HighCorrosion.label = ["high corrosion"]

class HighVibration(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(More) & hasParameter.some(Vibration))]

HighVibration.label = ["high vibration"]

class HighTemperature(Deviation):
    equivalent_to = [Deviation &
                     (hasGuideword.some(More) & hasParameter.some(Temperature))]

HighTemperature.label = ["high temperature"]

class LowTemperature(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(Less) & hasParameter.some(Temperature))]

LowTemperature.label = ["low temperature"]

class HighPressure(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(More) & hasParameter.some(Pressure))]

HighPressure.label = ["high pressure"]

class LowPressure(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(Less) & hasParameter.some(Pressure))]

LowPressure.label = ["low pressure"]

class NoFlow(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(No) & hasParameter.some(Flow))]

NoFlow.label = ["no flow"]

class HighFlow(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(More) & hasParameter.some(Flow))]

HighFlow.label = ["high flow"]

class LowFlow(Deviation):
    equivalent_to = [Deviation & (hasGuideword.only(Less) & hasParameter.only(Flow))]

LowFlow.label = ["low flow"]

class ElsewhereFlow(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(WhereElse) & hasParameter.some(Flow))]

ElsewhereFlow.label = ["elsewhere flow"]

class ReverseFlow(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(Reverse) & hasParameter.some(Flow))]

ReverseFlow.label = ["reverse flow"]

class OtherSequence(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(OtherThan) & hasParameter.some(Time))]

OtherSequence.label = ["other sequence"]

# === Also used for contamination
class OtherThanComposition(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(OtherThan) & hasParameter.some(Composition))]

OtherThanComposition.comment = ["Used for 'Contamination', 'Changed concentration', 'Different phase' etc."]
OtherThanComposition.label = ["other than composition"]

class HighLevel(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(More) & hasParameter.some(Level))]

HighLevel.label = ["high level"]

class LowLevel(Deviation):
    equivalent_to = [Deviation & (hasGuideword.some(Less) & hasParameter.some(Level))]

LowLevel.label = ["low level"]

AllDisjoint([HighVibration, HighTemperature, HighCorrosion, LowTemperature, HighCorrosion, HighPressure,
             LowPressure, NoFlow, HighFlow, LowFlow, ElsewhereFlow, OtherSequence, ReverseFlow,
             OtherThanComposition, HighLevel, LowLevel])
```

Appendix Listing H-1: Class definitions of deviation related concepts

Appendix I Formal ontology class definitions – equipment entity

```
# === HIGHER-LEVEL STRUCTURE
class PlantItem(Thing):
    pass

class FunctionalPlantItem(PlantItem):
    pass

class StructuralPlantItem(PlantItem):
    pass

AllDisjoint([FunctionalPlantItem, StructuralPlantItem])

# === Ports
class Port(Thing):
    pass

class ConnectionType(Thing):
    pass

class Inlet(ConnectionType):
    pass

class Outlet(ConnectionType):
    pass

class hasConnectionType(Port >> ConnectionType, FunctionalProperty):
    pass

class hasName(Port >> str, FunctionalProperty):
    pass

class portEquippedWithInstrumentation(Port >> PlantItem, FunctionalProperty):
    pass

# === Control instance
class ControlInstance(Thing):
    pass

class Operator(ControlInstance):
    pass

class ProgrammableLogicController(ControlInstance):
    pass

class OperatorAndProcessControlSystem(ControlInstance):
    pass

class NotControlled(ControlInstance):
    pass

AllDisjoint([Operator, ProgrammableLogicController, NotControlled])

# === FailSafePosition
class FailSafePosition(Thing):
    pass

class FailOpen(FailSafePosition):
    pass

class FailClosed(FailSafePosition):
    pass

# === OperatingConditions
class OperationMode(Thing):
    pass

class NormalOperation(OperationMode):
    pass

class StartUpOperation(OperationMode):
    pass

class ShutDownOperation(OperationMode):
    pass

class Maintenance(OperationMode):
    pass

# === Piping
class Piping(StructuralPlantItem):
    pass

class Pipe(Piping):
    pass

class BlanketingGasline(Pipe):
    pass

class TubeCoil(Pipe):
    pass

class TubeBundle(Pipe):
    pass

class VentPipe(Pipe):
    pass

class TankTruckHose(Pipe):
    pass

class Fitting(Piping):
    pass

# === Material transfer equipment
class MaterialTransferEquipment(FunctionalPlantItem):
    pass

class Pump(MaterialTransferEquipment):
    pass

class Fan(MaterialTransferEquipment):
    pass

class CentrifugalPump(Pump):
    pass
```

Appendix Listing I-1: Class definitions of equipment entity related concepts I

```
class ReciprocatingPump(Pump):
    pass

ReciprocatingPump.comment = ["is a positive displacement pump"]

class VacuumPump(Pump):
    pass

class Compressor(MaterialTransferEquipment):
    pass

class ScrewCompressor(Compressor):
    pass

class PistonCompressor(Compressor):
    pass

PistonCompressor.comment = ["piston compressor", "positive-displacement compressor"]

# ==== APPARATUS
class Apparatus(StructuralPlantItem):
    pass

class NoApparatus(Apparatus):
    pass

class AtmosphericStorageTank(Apparatus):
    pass

class PressureVessel(Apparatus):
    pass

class OpenVessel(Apparatus):
    pass

OpenVessel.comment = ["e.g. cooling tower"]

class Casing(Apparatus):
    pass

class Body(Apparatus):
    pass

Body.comment = ["e.g. valve body"]

class PumpCasing(Casing):
    pass

class CompressorCasing(Casing):
    pass

# === INSTRUMENTATION
class Instrumentation(FunctionalPlantItem):
    pass

# Trivial case
class NoInstrumentation(Instrumentation):
    pass

class FrequencyConverter(Instrumentation):
    pass

FrequencyConverter.comment = ["speed control pump/compressor"]

class Controller(Instrumentation):
    pass

class SpeedController(Controller):
    pass

class LevelIndicatorController(Controller):
    pass

class QualityIndicatorController(Controller):
    pass

class LevelIndicator(Instrumentation):
    pass

class PressureIndicatorController(Controller):
    pass

class FlowIndicatorController(Controller):
    pass

class Transmitter(Instrumentation):
    pass

class MonitoringSystem(Instrumentation):
    pass

class Alarm(MonitoringSystem):
    pass

class HighLevelAlarm(MonitoringSystem):
    pass

class FlashingLight(MonitoringSystem):
    pass

class Actuator(Instrumentation):
    pass

class ElectricalActuator(Actuator):
    pass

class ElectricMotor(ElectricalActuator):
    pass

class Solenoid(ElectricalActuator):
    pass

class HydraulicActuator(Actuator):
    pass

class PneumaticActuator(Actuator):
    pass

class ManualActuator(Actuator):
    pass

# === FIXTURE
class Fixture(StructuralPlantItem):
    pass
```

Appendix Listing I-2: Class definitions of equipment entity related concepts II

```
class NoFixture(Fixture):
    pass

class Jacket(Fixture):
    pass

class Tray(Fixture):
    pass

class ChimneyTray(Fixture):
    pass

class LiquidDistributor(Fixture):
    pass

LiquidDistributor.comment = ["Used in Cooling tower or wet scrubber", "can be spray system etc."]

class PackedBed(Fixture):
    pass

PackedBed.comment = ["Fill / Package / Fill Material"]

class Baffle(Fixture):
    pass

class Basin(Fixture):
    pass

class PlatePackage(Fixture):
    pass

PlatePackage.comment = ["For plate heat exchanger"]

class HalfPipeCoilJacket(Fixture):
    pass

class FinnedCoil(Fixture):
    pass

class Impeller(Fixture):
    pass

class Stirrer(Fixture):
    pass

# === OPERATION RELATED EQUIPMENT
class Subunit(FunctionalPlantItem):
    pass

class SealingSystem(Subunit):
    pass

SealingSystem.comment = [
    "[Seals] is a generic term for 'mech. seals', 'gasket', 'shaft seal', 'rotary seal', 'o-ring seal', "
    "'liquid seal'", "gasket: between flat flanges"]

class LubricationSystem(Subunit):
    pass

class NoLubricationSystem(Subunit):
    pass

class CoolingSystem(Subunit):
    pass

class HeatingSystem(Subunit):
    pass

class Burner(Subunit):
    pass

class Bypass(Subunit):
    pass

class ElectricalEnergySupply(Subunit):
    pass

class CompressedAirSupply(Subunit):
    pass

class SteamSupply(Subunit):
    pass

class CondensateSeparator(Subunit):
    pass

class InertgasSupply(Subunit):
    pass

class PhysicalDevice(Instrumentation):
    pass

class FlowControlValve(PhysicalDevice):
    pass

class PressureControlValve(PhysicalDevice):
    pass

class NonReturnValve(PhysicalDevice):
    pass

class ApiAdaptorValve(PhysicalDevice):
    pass
comment = ["https://www.opwglobal.com/products/us/transportation-products/"
           "tank-truck-products/mechanical-tank-truck-products/"
           "bottom-loading-adapters-gravity-couplers-dust-caps/api-adaptors"]

class ShutOffValve(PhysicalDevice):
    pass

class BottomDrainValve(PhysicalDevice):
    pass

class ThreeWayValve(PhysicalDevice):
    pass

class ThrottlingValve(PhysicalDevice):
    pass

class InletValve(PhysicalDevice):
    pass

class OutletValve(PhysicalDevice):
    pass

class Orifice(PhysicalDevice):
    pass
```

Appendix Listing I-3: Class definitions of equipment entity related concepts III

```
class EquipmentEntity(Thing):
    pass
EquipmentEntity.comment = ["Process unit is composed of plant items (fixture, instrumentation, support system)",
                           "has a nominal function and an operating state"]
class SourceEntity(EquipmentEntity):
    pass

class SinkEntity(EquipmentEntity):
    pass

class ConnectionPipeEntity(EquipmentEntity):
    pass

class TankTruckEntity(EquipmentEntity):
    pass

class StorageTankEntity(EquipmentEntity):
    pass

class SettlingTankEntity(EquipmentEntity):
    pass

class StabilizerColumnEntity(EquipmentEntity):
    pass

class DistillationColumnEntity(EquipmentEntity):
    pass

class SteamDrivenReboilerEntity(EquipmentEntity):
    pass

class PumpEntity(EquipmentEntity):
    pass

class CompressorEntity(EquipmentEntity):
    pass

class ValveEntity(EquipmentEntity):
    pass

class ReactorEntity(EquipmentEntity):
    pass

class WetScrubberEntity(EquipmentEntity):
    pass

class InertgasBlanketingEntity(EquipmentEntity):
    pass
# Pressure vessels
class PressureVesselEntity(EquipmentEntity):
    pass

class PressureReceiverEntity(PressureVesselEntity):
    pass

class SteamReceiverEntity(PressureVesselEntity):
    pass

class ShellTubeHeatExchangerEntity(EquipmentEntity):
    pass

class ShellTubeEvaporatorEntity(EquipmentEntity):
    pass

class PlateHeatExchangerEntity(EquipmentEntity):
    pass

class CoolingTowerEntity(EquipmentEntity):
    pass

class AirCooledCondenserEntity(EquipmentEntity):
    pass

class FinTubeEvaporatorEntity(EquipmentEntity):
    pass
# === Relations
class hasFixture(EquipmentEntity >> Fixture):
    pass

class isTransportable(EquipmentEntity >> bool, FunctionalProperty):
    pass

class hasConnectionToAdjacentPlantItem(EquipmentEntity >> PlantItem):
    pass

class hasInstrumentation(EquipmentEntity >> Instrumentation):
    pass

class hasFailSafePosition(EquipmentEntity >> FailSafePosition):
    pass

class hasMaterialTransferEquipment(EquipmentEntity >> MaterialTransferEquipment):
    pass

class hasSubunit(EquipmentEntity >> Subunit):
    pass

class hasApparatus(EquipmentEntity >> Apparatus):
    pass

class hasPiping(EquipmentEntity >> Piping):
    pass

class hasIdentifier(EquipmentEntity >> str):
    pass

class hasIntendedFunction(EquipmentEntity >> process_onto.IntendedFunction):
    pass

class hasMaximumOperatingPressureInBarGauge(EquipmentEntity >> float, FunctionalProperty):
    pass

class hasMaximumOperatingTemperatureInKelvin(EquipmentEntity >> float, FunctionalProperty):
    pass

class entityControlledBy(EquipmentEntity >> ControlInstance):
    pass

class hasOperationMode(EquipmentEntity >> OperationMode):
    pass

class formsControlLoopWith(Instrumentation >> Instrumentation):
    pass

class hasPort(EquipmentEntity >> Port):
    pass

class portEquippedWithInstrumentation(Port >> Instrumentation, FunctionalProperty):
    pass
```

Appendix Listing I-4: Class definitions of equipment entity related concepts IV

```
# === Intended function
class IntendedFunction(Thing):
    pass

class NoIntendedFunction(IntendedFunction):
    pass

class Evaporating(IntendedFunction):
    pass

class Condensing(IntendedFunction):
    pass

class HeatTransferring(IntendedFunction):
    pass

class Mixing(IntendedFunction):
    pass

class Separating(IntendedFunction):
    pass

class MaterialTransfer(IntendedFunction):
    pass

class Stabilizing(IntendedFunction):
    pass

class FlowControl(IntendedFunction):
    pass

class PressureControl(IntendedFunction):
    pass

class Reacting(IntendedFunction):
    pass

class Purifying(IntendedFunction):
    pass

class Inerting(IntendedFunction):
    pass

class Transporting(IntendedFunction):
    pass

class Loading(IntendedFunction):
    pass

class Unloading(IntendedFunction):
    pass

class Emptying(IntendedFunction):
    pass

class Filling(IntendedFunction):
    pass

class Storing(IntendedFunction):
    pass

class ModeIndependent(IntendedFunction):
    pass

class DeliverConstantVolumeFlow(IntendedFunction):
    pass
```

Appendix Listing I-5: Class definitions of intended functions

Appendix J Formal ontology class definitions substance

```
class Substance(Thing):
    pass

class Property(Thing):
    pass

class StabilityReactivityInformation(Thing):
    pass

class hasStabilityReactivityInformation(Substance >> StabilityReactivityInformation):
    pass

class ReactsViolentlyWithOxidizer(StabilityReactivityInformation):
    pass

class FormsExplosiveMixtureWithAir(StabilityReactivityInformation):
    pass

class FormsExplosiveMixturesWithOxidizingAgents(StabilityReactivityInformation):
    pass

class ReactsWithWater(StabilityReactivityInformation):
    pass

class ReactsWithChlorates(StabilityReactivityInformation):
    pass

class PolymerizesExothermicallyWithoutInhibitor(StabilityReactivityInformation):
    pass

class PolymerizesExothermicallyWhenExposedToLight(StabilityReactivityInformation):
    pass

class PolymerizesExothermicallyWhenExposedToHeat(StabilityReactivityInformation):
    pass

class FormationOfHazardousDecompositionProducts(StabilityReactivityInformation):
    pass

class ThermalDecompositionGeneratesCorrosiveVapors(FormationOfHazardousDecompositionProducts):
    pass

class IncompatibleToStrongAcids(StabilityReactivityInformation):
    pass

class IncompatibleToStrongBases(StabilityReactivityInformation):
    pass

class IncompatibleToStrongOxidizers(StabilityReactivityInformation):
    pass

class SpecificSubstanceTask(Thing):
    pass

class hasSpecificTask(Substance >> SpecificSubstanceTask):
    pass

class ScrubbingAgent(SpecificSubstanceTask):
    pass

class Stabilizer(SpecificSubstanceTask):
    pass

Stabilizer.comment = ["chemical that is used to prevent degradation"]

class ReactionInhibitor(SpecificSubstanceTask):
    pass

ReactionInhibitor.comment = ["Substance that decreases or prevents chemical reaction"]

class Lubricant(SpecificSubstanceTask):
    pass

class Refrigerant(SpecificSubstanceTask):
    pass

class ProcessMedium(SpecificSubstanceTask):
    pass

class CoolingMedium(SpecificSubstanceTask):
    pass

class HeatingMedium(SpecificSubstanceTask):
    pass

class InertGas(SpecificSubstanceTask):
    pass

# Source 1: https://www.chemsafetypro.com/Topics/GHS/GHS_Classification_Criteria.html, Source 2: https://pubchem.ncbi.nlm.nih.gov/ghs/
class HazardClass(Thing):
    pass

class hasHazardClass(Substance >> HazardClass):
    pass

class PhysicalHazard(HazardClass):
    pass

class HealthHazard(HazardClass):
    pass

class EnvironmentalHazard(HazardClass):
    pass

class Explosives(PhysicalHazard):
    pass

class FlammableGases(PhysicalHazard):
    pass

class FlammableGasCategory1(FlammableGases):
    pass

class FlammableGasCategory2(FlammableGases):
    pass

class PyrophoricGasCategory1(FlammableGases):
    pass
```

Appendix Listing J-1: Class definitions of substance related concepts Part I

```
class ChemicallyUnstableGasCategoryA(FlammableGases):
    pass

class ChemicallyUnstableGasCategoryB(FlammableGases):
    pass

class Aerosols(PhysicalHazard):
    pass

class AerosolCategory1(Aerosols):
    pass

class AerosolCategory2(Aerosols):
    pass

class OxidizingGases(PhysicalHazard):
    pass

class GasesUnderPressure(PhysicalHazard):
    pass

class CompressedGas(GasesUnderPressure):
    pass

class LiquefiedGas(GasesUnderPressure):
    pass

class RefrigeratedLiquefiedGas(GasesUnderPressure):
    pass

class DissolvedGas(GasesUnderPressure):
    pass

class FlammableLiquids(PhysicalHazard):
    pass

class FlammableLiquidCategory1(FlammableLiquids):
    pass

class FlammableLiquidCategory2(FlammableLiquids):
    pass

class FlammableLiquidCategory3(FlammableLiquids):
    pass

class FlammableLiquidCategory4(FlammableLiquids):
    pass

class FlammableSolids(PhysicalHazard):
    pass

class SelfReactiveSubstances(PhysicalHazard):
    pass

class PyrophoricLiquids(PhysicalHazard):
    pass

class PyrophoricSolids(PhysicalHazard):
    pass

class SelfHeatingSubstances(PhysicalHazard):
    pass

class EmitFlammableGasesWithWater(PhysicalHazard):
    pass

class OxidizingLiquids(PhysicalHazard):
    pass

class OxidizingSolids(PhysicalHazard):
    pass

class OrganicPeroxides(PhysicalHazard):
    pass

class CorrosiveToMetals(PhysicalHazard):
    pass

class DesensitiziedExplosives(PhysicalHazard):
    pass

class HazardousToAquaticEnvironment(EnvironmentalHazard):
    pass

class HazardousToAquaticEnvironmentLongTermCategory1(HazardousToAquaticEnvironment):
    pass

class HazardousToAquaticEnvironmentLongTermCategory2(HazardousToAquaticEnvironment):
    pass

class HazardousToAquaticEnvironmentLongTermCategory3(HazardousToAquaticEnvironment):
    pass

class HazardousToAquaticEnvironmentLongTermCategory4(HazardousToAquaticEnvironment):
    pass

class HazardousToAquaticEnvironmentAcuteCategory1(HazardousToAquaticEnvironment):
    pass

class HazardousToAquaticEnvironmentAcuteCategory2(HazardousToAquaticEnvironment):
    pass

class HazardousToAquaticEnvironmentAcuteCategory3(HazardousToAquaticEnvironment):
    pass

class HazardousToOzoneLayer(EnvironmentalHazard):
    pass

class AcuteToxicity(HealthHazard):
    pass

class AcuteToxicityCategory1(AcuteToxicity):
    pass

class AcuteToxicityCategory2(AcuteToxicity):
    pass

class AcuteToxicityCategory3(AcuteToxicity):
    pass
```

Appendix Listing J-2: Class definitions of substance related concepts Part II

```
class SpecificTargetOrganToxicitySingleExposure(HealthHazard):
    pass

class SpecificTargetOrganToxicityRepeatedExposure(HealthHazard):
    pass

class SpecificTargetOrganToxicitySingleExposureCategory1(HealthHazard):
    pass

class SpecificTargetOrganToxicitySingleExposureCategory2(HealthHazard):
    pass

class SpecificTargetOrganToxicitySingleExposureCategory3(HealthHazard):
    pass

class SpecificTargetOrganToxicityRepeatedExposureCategory1(HealthHazard):
    pass

class SpecificTargetOrganToxicityRepeatedExposureCategory2(HealthHazard):
    pass

class SpecificTargetOrganToxicityRepeatedExposureCategory3(HealthHazard):
    pass

class SkinCorrosionIrritation(HealthHazard):
    pass

class SkinCorrosionIrritationCategory1(SkinCorrosionIrritation):
    pass

class SkinCorrosionIrritationCategory2(SkinCorrosionIrritation):
    pass

class SkinCorrosionIrritationCategory3(SkinCorrosionIrritation):
    pass

class SeriousEyeDamageIrritation(HealthHazard):
    pass

class SeriousEyeDamageIrritationCategory1(HealthHazard):
    pass

class SeriousEyeDamageIrritationCategory2A(HealthHazard):
    pass

class SeriousEyeDamageIrritationCategory2B(HealthHazard):
    pass

class RespiratoryOrSkinSensitization(HealthHazard):
    pass

class GermCellMutagenicity(HealthHazard):
    pass

class Carcinogenicity(HealthHazard):
    pass

class ReproductiveToxicology(HealthHazard):
    pass

class ReproductiveToxicityCategory1(HealthHazard):
    pass

class ReproductiveToxicityCategory2(HealthHazard):
    pass

class TargetOrganSystemicToxicitySingleExposure(HealthHazard):
    pass

class TargetOrganSystemicToxicityRepeatedExposure(HealthHazard):
    pass

class AspirationToxicity(HealthHazard):
    pass

class AspirationHazardCategory1(AspirationToxicity):
    pass

class AspirationHazardCategory2(AspirationToxicity):
    pass

class Flashpoint(Property):
    pass

Flashpoint.comment = ["Flammpunkt"]

class hasFlashpointInKelvin(Substance >> float, FunctionalProperty):
    pass

class hasFreezingPointInKelvin(Substance >> float, FunctionalProperty):
    pass

hasFreezingPointInKelvin.comment = ["Schmelztemperatur gleich groß wie Erstarrungstemperatur"]

class hasVaporPressureInPascal(Substance >> float, FunctionalProperty):
    pass

class hasUpperExplosionLimitInPercent(Substance >> float, FunctionalProperty):
    pass

class hasLowerExplosionLimitInPercent(Substance >> float, FunctionalProperty):
    pass

class hasAutoIgnitionTemperatureInKelvin(Substance >> float, FunctionalProperty):
    pass

class hasBoilingPointInKelvin(Substance >> float, FunctionalProperty):
    pass

class StateOfAggregation(Substance):
    pass

class Liquid(StateOfAggregation):
    pass

class Gaseous(StateOfAggregation):
    pass

class Multiphase(StateOfAggregation):
    pass

class hasStateOfAggregation(Substance >> StateOfAggregation):
    pass
```

Appendix Listing J-3: Class definitions of substance related concepts Part III

Appendix K Formal ontology class definitions underlying causes

```
class UtilityFailure(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(NoInertgasSupply | NoSteamFlow)
                     ]

class MalfunctionUpstreamProcess(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(DeliveryOfHighVolatilityComponents)
                     ]

class IntroductionOfRainwater(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ContaminationInUnloadingLines) &
                      underlyingcauseRequiresBoundaryCondition.some(boundary_onto.LocatedOutside))
                     ]

class MalfunctionFlowController(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ExcessiveInflow |
                                                    LossOfInflow) &
                      underlyingcauseInvolvesEquipmentEntity.some(
                          equipment_onto.hasInstrumentation.some(equipment_onto.FlowControlValve) |
                                                                 equipment_onto.hasInstrumentation.some(
                                                                     equipment_onto.LevelIndicatorController |
                                                                     equipment_onto.FlowIndicatorController)))]

class MalfunctionPressureController(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(IncreasedInletPressure |
                                                    IncorrectPressureAdjustment) &
                      underlyingcauseInvolvesEquipmentEntity.some(
                          equipment_onto.hasInstrumentation.some(equipment_onto.PressureControlValve)))]

class AmbientTemperatureChange(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(InsufficientThermalInbreathing)]

class VehicleCollision(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(PhysicalImpact)]

class IncorrectCrossConnection(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(AbnormalVaporIntake)]

class ImproperProcessHeatInput(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ThermalExpansion) &
                      underlyingcauseInvolvesEquipmentEntity.some(
                          equipment_onto.hasSubunit.some(equipment_onto.HeatingSystem)))]

class FastGasRelaxation(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(AbruptReliefOfContent)]

class ExternalFire(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     ((isUnderlyingcauseOfCause.some(AbnormalHeatInput) &
                       underlyingcauseRequiresBoundaryCondition.some(boundary onto.ExternalFirePossible))
                      |
                      (underlyingcauseInvolvesEquipmentEntity.some(
                          equipment onto.hasApparatus.some(equipment onto.PressureVessel)) &
                       isUnderlyingcauseOfCause.some(ThermalExpansion) &
                       underlyingcauseRequiresBoundaryCondition.some(boundary onto.ExternalFirePossible)
                       )
                      |
                      (isUnderlyingcauseOfCause.some(AbnormalHeatInput) &
                       underlyingcauseInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                                   equipment_onto.hasIntendedFunction.some(
                                                                       process_onto.Storing)) &
                       underlyingcauseRequiresBoundaryCondition.some(boundary_onto.ExternalFirePossible)))
                     ]

class SolarRadiation(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     ((isUnderlyingcauseOfCause.some(AbnormalHeatInput) &
                       underlyingcauseInvolvesSubstance.some(
                           (substance_onto.hasFlashpointInKelvin <= 348.15) &
                           substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                       underlyingcauseRequiresBoundaryCondition.some(boundary_onto.LocatedOutside))
                      |
                      (isUnderlyingcauseOfCause.some(InsufficientThermalOutbreathing) &
                       underlyingcauseInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity) &
                       underlyingcauseRequiresBoundaryCondition.some(boundary onto.LocatedOutside))
                     )
                     ]

SolarRadiation.comment = ["Assumption behind surface temperature of tank/vessel can rise to 75 °C, flash point is compared to it"]

class RapidlyClosingValve(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(WaterHammer) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity))
                     ]

class BlockedPipingAndHeatInput(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ThermalExpansion) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity) &
                      underlyingcauseRequiresBoundaryCondition.some(boundary onto.ExternalFirePossible |
                                                                    boundary_onto.LocatedOutside)
                      )
                     ]

BlockedPipingAndHeatInput.comment = ["Requires external heat, therefore the boundary conditions"]

class AbnormallyHotIntake(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     ((isUnderlyingcauseOfCause.some(AbnormalHeatInput |
                                                     ThermalExpansion) &
                       underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.SteamReceiverEntity |
                                                                   equipment_onto.PressureReceiverEntity |
                                                                   equipment_onto.ReactorEntity))
                      |
                      (isUnderlyingcauseOfCause.some(AbnormalHeatInput) &
                       underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                                   equipment onto.hasIntendedFunction.some(
                                                                       process onto.Filling)))
                      )]
```

Appendix Listing K-1: Class definitions of underlying cause concepts Part I

```
class DepositionOfImpurities(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(BlockedInflowLine |
                                                   ReducedFlowArea)]

class LevelIndicatorControllerFailure(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ValveWronglyClosed |
                                                    ValveWronglyOpened |
                                                    IncorrectSetPointControlValve) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment onto.hasInstrumentation.some(equipment onto.LevelIndicatorController) |
                                                                  equipment onto.hasConnectionToAdjacentPlantItem.some(equipment onto.LevelIndicatorController)))]

class ControlValveFailsOpen(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ValveWronglyOpened) &
                      underlyingcauseInvolvesEquipmentEntity.some(
                          equipment onto.hasInstrumentation.some(equipment onto.FlowControlValve |
                                                                 equipment onto.PressureControlValve)))
                     ]

class ControlValveFailsClosed(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ValveWronglyClosed) &
                      underlyingcauseInvolvesEquipmentEntity.some(
                          equipment onto.hasInstrumentation.some(equipment onto.FlowControlValve |
                                                                 equipment onto.PressureControlValve)))
                     ]

class PressureIndicatorControllerFailure(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ValveWronglyClosed |
                                                    ValveWronglyOpened |
                                                    IncorrectPressureAdjustment |
                                                    IncorrectSetPointControlValve) &
                      underlyingcauseInvolvesEquipmentEntity.some(
                          equipment onto.hasInstrumentation.some(equipment onto.PressureIndicatorController) |
                          equipment onto.hasConnectionToAdjacentPlantItem.some(equipment onto.PressureIndicatorController)))
                     ]

class FlowIndicatorControllerFailure(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ValveWronglyClosed |
                                                    ValveWronglyOpened |
                                                    IncorrectSetPointControlValve) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment onto.SteamDrivenReboilerEntity &
                                                                  (equipment onto.hasInstrumentation.some(
                                                                      equipment onto.PressureIndicatorController) |
                                                                   equipment onto.hasConnectionToAdjacentPlantItem.some(
                                                                       equipment_onto.PressureIndicatorController)))
                      )]

class AbnormalHeatRemoval(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(FreezeUp)]

class LowAmbientTemperature(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (underlyingcauseRequiresBoundaryCondition.some(boundary onto.LocatedOutside) &
                      isUnderlyingcauseOfCause.some(FreezeUp))]

class DefectiveSeal(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.hasSubunit.some(equipment_onto.SealingSystem)) &
                      isUnderlyingcauseOfCause.some(ExternalLeakage))]

DefectiveSeal.comment = ["Called 'seal failure' in Lees' Loss Prevention ... pp. 12/40"]

class DepositionOfImpurities(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(BlockedOutflowLine)]

class MechanicalDamage(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(EquipmentFailure |
                                                   HeatInputByRecirculationPump)]

class WearDown(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(PumpSealFailure) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment onto.PumpEntity))
                     |
                     (isUnderlyingcauseOfCause.some(InternalLeakage) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment onto.SteamDrivenReboilerEntity))
                     ]

class LossOfLeakTightness(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(ExternalLeakage | LeakingDrainValve)
                     ]

class SuddenStartingPump(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (underlyingcauseInvolvesEquipmentEntity.some(
                         equipment onto.hasMaterialTransferEquipment.some(equipment onto.ReciprocatingPump)) &
                      isUnderlyingcauseOfCause.some(WaterHammer))]

class SuddenlyStoppingPump(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (underlyingcauseInvolvesEquipmentEntity.some(
                         equipment onto.hasMaterialTransferEquipment.some(equipment onto.ReciprocatingPump)) &
                      isUnderlyingcauseOfCause.some(WaterHammer))]

class PowerFailure(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     ((underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.hasSubunit.some(equipment_onto.ElectricalEnergySupply)) &
                       isUnderlyingcauseOfCause.some(EquipmentFailure))
                      |
                      (isUnderlyingcauseOfCause.some(WaterHammer) &
                       underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.ValveEntity &
                                                                   equipment onto.hasSubunit.some(equipment onto.ElectricalEnergySupply))
                       )
                      |
                      (isUnderlyingcauseOfCause.some(WaterHammer) &
                       underlyingcauseInvolvesEquipmentEntity.some(
                           (equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.ReciprocatingPump)) &
                           equipment_onto.hasSubunit.some(equipment_onto.ElectricalEnergySupply))
                       ))]

class MalfunctionSpeedControl(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (underlyingcauseInvolvesEquipmentEntity.some(
                         equipment onto.hasInstrumentation.some(equipment onto.SpeedController)) &
                      isUnderlyingcauseOfCause.some(WrongRotatingSpeed |
                                                    EquipmentFailure))]
```

Appendix Listing K-2: Class definitions of underlying cause concepts Part II

```
class BreakdownOfActuator(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (underlyingcauseInvolvesEquipmentEntity.some(
                         equipment_onto.hasInstrumentation.some(equipment_onto.ElectricalActuator |
                                                                equipment_onto.ManualActuator |
                                                                equipment_onto.HydraulicActuator |
                                                                equipment_onto.PneumaticActuator)) &
                      (isUnderlyingcauseOfCause.some(EquipmentFailure)))]

class FailureControlLoop(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     ((underlyingcauseInvolvesEquipmentEntity.some(equipment onto.hasInstrumentation.some(equipment onto.Controller) &
                                                                   equipment onto.entityControlledBy.some(equipment onto.ProgrammableLogicController)) &
                       isUnderlyingcauseOfCause.some(EquipmentFailure |
                                                     DeadHeadingOfPump))
                     |
                     (underlyingcauseInvolvesEquipmentEntity.some(equipment onto.hasInstrumentation.some(equipment onto.Controller) &
                                                                  equipment onto.entityControlledBy.some(equipment onto.ProgrammableLogicController)) &
                      isUnderlyingcauseOfCause.some(CoolingFailure))
                     |
                     isUnderlyingcauseOfCause.some(IncorrectIndicationOfFillingLevel)
                     |
                     isUnderlyingcauseOfCause.some(NoSteamFlow))]

class ValveFailure(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ClosedInletValve) |
                      isUnderlyingcauseOfCause.some(ClosedOutletValve) |
                      isUnderlyingcauseOfCause.some(OpenedInletValve) |
                      isUnderlyingcauseOfCause.some(OpenedOutletValve))]

class MaintenanceError(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(WrongMountingOfNonReturnValve | LeakingDrainValve | MissingImpeller)]

class OperationalError(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     ((underlyingcauseInvolvesEquipmentEntity.some(equipment onto.entityControlledBy.some(equipment onto.Operator)) &
                       isUnderlyingcauseOfCause.some(ConfusionOfSubstances | WrongTankLinedUp))
                      |
                      ((isUnderlyingcauseOfCause.some(ClosedInletValve) |
                        isUnderlyingcauseOfCause.some(ClosedOutletValve) |
                        isUnderlyingcauseOfCause.some(OpenedInletValve) |
                        isUnderlyingcauseOfCause.some(OpenedOutletValve) |
                        isUnderlyingcauseOfCause.some(PumpIncorrectlySet) |
                        isUnderlyingcauseOfCause.some(ValveClosedPressureBuildUpInPiping) |
                        isUnderlyingcauseOfCause.some(ExcessiveInflow) |
                        isUnderlyingcauseOfCause.some(DeadHeadingOfPump) |
                        isUnderlyingcauseOfCause.some(ConnectionsFaultyConnected) |
                        isUnderlyingcauseOfCause.some(BypassOpened) |
                        isUnderlyingcauseOfCause.some(ValveWronglyClosed) |
                        isUnderlyingcauseOfCause.some(ValveWronglyOpened) |
                        isUnderlyingcauseOfCause.some(IncorrectFilling) |
                        isUnderlyingcauseOfCause.some(ValveIntactUnintentionallyClosed) |
                        isUnderlyingcauseOfCause.some(DrainValveInadvertentlyOpened) |
                        isUnderlyingcauseOfCause.some(IncorrectSetPointControlValve)
                        )))]

class ContaminationInTankTruck(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     (underlyingcauseInvolvesEquipmentEntity.some(equipment_onto.TankTruckEntity) &
                      isUnderlyingcauseOfCause.some(InadvertentContamination))]

class CondensationAirHumidity(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(ContaminationByWater) &
                     underlyingcauseInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                                 equipment onto.hasIntendedFunction.some(process onto.Storing))]

class EntryDuringFilling(UnderlyingCause):
    equivalent_to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(ContaminationByWater) &
                     underlyingcauseInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                                 equipment onto.hasIntendedFunction.some(process onto.Filling))]

class LongStorageTimeOfStabilizer(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(TooLittleStabilizer)]

class PersistentMechanicalStresses(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     isUnderlyingcauseOfCause.some(MaterialDegradation)]

class HoseIncorrectlyConnected(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                      (isUnderlyingcauseOfCause.some(ExternalLeakage) &
                       underlyingcauseInvolvesEquipmentEntity.some(
                           equipment_onto.hasPiping.some(equipment_onto.TankTruckHose) &
                           equipment_onto.entityControlledBy.some(equipment_onto.Operator)))]

class BrokenHose(UnderlyingCause):
    equivalent to = [UnderlyingCause &
                     (isUnderlyingcauseOfCause.some(ExternalLeakage) &
                      underlyingcauseInvolvesEquipmentEntity.some(equipment onto.hasPiping.some(equipment onto.TankTruckHose)))]

class MalfunctionOilTemperatureControl(causes_onto.UnderlyingCause):
    equivalent_to = [causes_onto.UnderlyingCause &
                     causes_onto.isUnderlyingcauseOfCause.some(MalfunctionLubricationSystem)]

class FailureOilCoolingFan(causes_onto.UnderlyingCause):
    equivalent to = [causes onto.UnderlyingCause &
                     causes onto.isUnderlyingcauseOfCause.some(MalfunctionLubricationSystem)]

class CloggingOilCoolingLine(causes onto.UnderlyingCause):
    equivalent_to = [causes_onto.UnderlyingCause &
                     causes_onto.isUnderlyingcauseOfCause.some(MalfunctionLubricationSystem)]

class CloggingOilFilter(causes_onto.UnderlyingCause):
    equivalent_to = [causes_onto.UnderlyingCause &
                     causes_onto.isUnderlyingcauseOfCause.some(MalfunctionLubricationSystem)]

class LowCompressorOilLevel(causes onto.UnderlyingCause):
    equivalent to = [causes onto.UnderlyingCause &
                     causes onto.isUnderlyingcauseOfCause.some(MalfunctionLubricationSystem)]

class EntryOfForeignGases(causes_onto.UnderlyingCause):
    equivalent_to = [causes_onto.UnderlyingCause &
                     (causes_onto.isUnderlyingcauseOfCause.some(causes_onto.NonCondensables) &
                      causes_onto.underlyingcauseRequiresBoundaryCondition.some(boundary_onto.IntroductionOfAir))]

class Sediments(causes onto.UnderlyingCause):
    equivalent to = [causes onto.UnderlyingCause &
                     (causes onto.isUnderlyingcauseOfCause.some(causes onto.Fouling) &
                      causes_onto.underlyingcauseRequiresBoundaryCondition.some(boundary_onto.IntroductionOfImpurities))]
```

Appendix Listing K-3: Class definitions of underlying cause concepts Part III

```
class GrowthOfOrganisms(causes onto.UnderlyingCause):
    equivalent to = [causes onto.UnderlyingCause &
                     (causes onto.isUnderlyingcauseOfCause.some(causes onto.Fouling)
                      )]

class MalfunctionControlAir(causes_onto.UnderlyingCause):
    equivalent_to = [causes_onto.UnderlyingCause &
                     (causes_onto.isUnderlyingcauseOfCause.some(causes_onto.ValveWronglyClosed |
                                                                causes onto.ValveWronglyOpened |
                                                                causes onto.ValveClosedPressureBuildUpInPiping |
                                                                FailureOfControlSystem) &
                      causes onto.underlyingcauseInvolvesEquipmentEntity.some(equipment onto.ValveEntity &
                                                                              equipment onto.hasSubunit.some(equipment onto.CompressedAirSupply) &
                                                                              equipment_onto.hasInstrumentation.some(equipment_onto.PneumaticActuator)))]


class WrongElectricSignal(causes onto.UnderlyingCause):
    equivalent to = [causes onto.UnderlyingCause &
                     (causes onto.isUnderlyingcauseOfCause.some(causes onto.ValveWronglyClosed |
                                                                causes onto.ValveWronglyOpened |
                                                                causes onto.ValveClosedPressureBuildUpInPiping |
                                                                FailureOfControlSystem) &
                      causes onto.underlyingcauseInvolvesEquipmentEntity.some(equipment onto.ValveEntity &
                                                                              equipment_onto.hasSubunit.some(equipment_onto.ElectricalEnergySupply) &
                                                                              equipment_onto.hasInstrumentation.some(equipment_onto.ElectricalActuator)
                                                                              ))]
```

Appendix Listing K-4: Class definitions of underlying cause concepts Part IV

Appendix L Formal ontology class definitions causes

```
class ReducedFlowArea(Cause):
    equivalent to = [Cause &
                     (causeInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity) &
                      isCauseOfDeviation.some(deviation onto.LowFlow))]

class HosePipeBlocked(Cause):
    equivalent to = [Cause &
                     (isCauseOfDeviation.some(deviation onto.NoFlow) &
                      causeInvolvesEquipmentEntity.some(equipment onto.TankTruckEntity &
                                                        equipment onto.hasPiping.some(equipment onto.TankTruckHose)))]

class PhysicalImpact(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation_onto.ElsewhereFlow) &
                      causeInvolvesSiteInformation.some(
                          site_information.involvesPlantAmbientInformation.some(site_information.VehicleTraffic |
                                                                                site information.CranePresent)) &
                      causeInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                        equipment onto.hasIntendedFunction.some(
                                                            process onto.ModeIndependent)))]

class MechanicalFailureOfSupport(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation_onto.ElsewhereFlow) &
                      causeRequiresBoundaryCondition.some(boundary_onto.FoundationCanBeAffected) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                        equipment onto.hasIntendedFunction.some(
                                                            process onto.ModeIndependent)))]

class Pollution(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation_onto.OtherThanComposition) &
                      causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfImpurities) &
                      causeInvolvesEquipmentEntity.some(
                          equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank |
                                                           equipment_onto.PressureVessel)))]

class NoFeed(Cause):
    equivalent to = [Cause &
                     (isCauseOfDeviation.some(deviation onto.NoFlow) &
                      causeInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.StabilizerColumnEntity |
                                                        equipment_onto.ReactorEntity))]

class WrongMountingOfNonReturnValve(Cause):
    equivalent to = [Cause &
                     (isCauseOfDeviation.some(deviation onto.NoFlow) &
                      causeInvolvesEquipmentEntity.some(equipment onto.hasInstrumentation.some(equipment onto.NonReturnValve))
                      )]

# Unit tests conducted @200414 [TestFreezeUp] in unit_tests\overarching_phenomena.py
class FreezeUp(Cause):
    equivalent to = [Cause &
                     (causeInvolvesSubstance.some(
                         substance onto.hasStateOfAggregation.some(substance onto.Liquid) &
                         (substance onto.hasFreezingPointInKelvin >= upper onto.lowest ambient temperature)
                         )
                      &
                      causeInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity) &
                      isCauseOfDeviation.some(deviation_onto.LowTemperature))]

FreezeUp.comment = ["[FreezeUp] is seen as a cause for no flow.",
                    "It can also be modeled as an [Effect] (T_low & pipe -> [FreezeUp])",
                    "The modelling is consequence-oriented, thus (T_low & pipe -> [PipeFracture])"]

class ValveClosedPressureBuildUpInPiping(Cause):
    equivalent to = [Cause &
                     (causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasInstrumentation.some(equipment_onto.ShutOffValve) &
                         equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                      isCauseOfDeviation.some(deviation_onto.HighPressure))
                     |
                     (causeInvolvesEquipmentEntity.some(
                         equipment onto.hasInstrumentation.some(equipment onto.FlowControlValve) &
                         equipment onto.hasIntendedFunction.some(process onto.FlowControl)) &
                      isCauseOfDeviation.some(deviation onto.HighPressure))
                     |
                     (causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasInstrumentation.some(equipment_onto.PressureControlValve) &
                         equipment_onto.hasIntendedFunction.some(process_onto.PressureControl)) &
                      isCauseOfDeviation.some(deviation_onto.HighPressure))]

class DrainValveInadvertentlyOpened(Cause):
    equivalent to = [Cause &
                     ((causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasInstrumentation.some(equipment_onto.BottomDrainValve) &
                         equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                       causeInvolvesSubstance.some(substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                                             substance_onto.Multiphase)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow))
                      |
                      (causeInvolvesEquipmentEntity.some(
                          equipment onto.hasInstrumentation.some(equipment onto.BottomDrainValve) &
                          equipment onto.hasIntendedFunction.some(process onto.DeliverConstantVolumeFlow |
                                                                  equipment onto.NormalOperation)) &
                       causeInvolvesSubstance.some(substance_onto.hasStateOfAggregation.some(substance_onto.Liquid)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow)))]

class LiquidTransferWithoutCompensation(Cause):
    equivalent_to = [Cause &
                     (causeInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                        equipment onto.hasIntendedFunction.some(
                                                                                            process onto.Emptying)) &
                      causeInvolvesSubstance.some(substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                                            substance_onto.Multiphase)) &
                      isCauseOfDeviation.some(deviation_onto.LowPressure))
                     ]

class InsufficientThermalInbreathing(Cause):
    equivalent to = [Cause &
                     causeInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                       equipment onto.hasIntendedFunction.some(
                                                                                           process onto.Storing)) &
                     isCauseOfDeviation.some(deviation onto.LowPressure)
                     ]
```

Appendix Listing L-1: Class definitions of cause concepts Part I

```
class InsufficientThermalOutbreathing(Cause):
    equivalent_to = [Cause &
                     causeInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                       equipment_onto.hasIntendedFunction.some(
                                                                                              process_onto.Storing)) &
                     isCauseOfDeviation.some(deviation_onto.HighPressure)]

class NoInertgasSupply(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation onto.NoFlow) &
                      causeInvolvesEquipmentEntity.some(equipment onto.InertgasBlanketingEntity &
                                                        equipment onto.hasIntendedFunction.some(
                                                                                               process onto.Inerting)) &
                      causeInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.InertGas)))]

class IncreasedInletPressure(Cause):
    equivalent_to = [Cause &
                     ((isCauseOfDeviation.some(deviation onto.HighFlow) &
                       causeInvolvesEquipmentEntity.some(equipment onto.InertgasBlanketingEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                                                                process onto.Inerting)) &
                       causeInvolvesSubstance.some(substance onto.hasSpecificTask.some(substance onto.InertGas))))]

class IncorrectPressureAdjustment(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation onto.LowPressure) &
                      causeInvolvesEquipmentEntity.some(
                          equipment onto.hasIntendedFunction.some(process onto.ModeIndependent) &
                          equipment onto.hasPiping.some(equipment onto.BlanketingGasline) &
                          equipment onto.hasInstrumentation.some(equipment onto.PressureIndicatorController)))]

class BlockedOutflowLine(Cause):
    equivalent_to = [Cause &
                     (
                     (causeInvolvesEquipmentEntity.some(equipment_onto.WetScrubberEntity |
                                                        equipment onto.AirCooledCondenserEntity |
                                                        equipment onto.ShellTubeEvaporatorEntity |
                                                        equipment onto.FinTubeEvaporatorEntity) &
                      isCauseOfDeviation.some(deviation onto.NoFlow |
                                              deviation_onto.HighPressure))
                     |
                     (causeInvolvesEquipmentEntity.some(equipment_onto.CompressorEntity) &
                      isCauseOfDeviation.some(deviation_onto.NoFlow |
                                              deviation_onto.HighPressure) &
                      causeInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.ProcessMedium |
                                                              substance onto.Refrigerant)))
                     |
                     (causeInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity) &
                      isCauseOfDeviation.some(deviation_onto.HighLevel) &
                      causeInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium))))]

class TooLittleStabilizer(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation onto.OtherThanComposition) &
                      causeInvolvesEquipmentEntity.some((equipment onto.ConnectionPipeEntity |
                                                         equipment onto.StorageTankEntity |
                                                         equipment onto.SettlingTankEntity |
                                                         equipment_onto.PressureVesselEntity |
                                                         equipment_onto.WetScrubberEntity |
                                                         equipment_onto.PressureReceiverEntity |
                                                         equipment_onto.ShellTubeHeatExchangerEntity |
                                                         equipment_onto.ShellTubeEvaporatorEntity |
                                                         equipment onto.FinTubeEvaporatorEntity)) &
                      causeRequiresBoundaryCondition.some(boundary onto.SubstanceContainsStabilizer) &
                      causeInvolvesSubstance.some(
                          substance onto.hasStabilityReactivityInformation.some(substance onto.PolymerizesExothermicallyWithoutInhibitor)))]

class TooLittleInhibitor(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation_onto.OtherThanComposition) &
                      causeInvolvesEquipmentEntity.some((equipment onto.ConnectionPipeEntity |
                                                         equipment onto.StorageTankEntity |
                                                         equipment onto.SettlingTankEntity |
                                                         equipment_onto.PressureVesselEntity |
                                                         equipment_onto.WetScrubberEntity |
                                                         equipment_onto.PressureReceiverEntity |
                                                         equipment_onto.ShellTubeHeatExchangerEntity |
                                                         equipment_onto.ShellTubeEvaporatorEntity |
                                                         equipment onto.FinTubeEvaporatorEntity)) &
                      causeInvolvesSubstance.some(
                          substance onto.hasStabilityReactivityInformation.some(substance onto.PolymerizesExothermicallyWithoutInhibitor)))]

class ExcessiveFluidWithdrawal(Cause):
    equivalent_to = [Cause &
                     causeInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity) &
                     isCauseOfDeviation.some(deviation_onto.LowPressure)]

ExcessiveFluidWithdrawal.comment = ["The issue of depressurization and collapse of vessel is addressed",
                                    "https://www.aiche.org/resources/publications/cep/2019/december/protect-tanks-overpressure-and-vacuum",
                                    "API 2000"]

class WaterHammer(Cause):
    equivalent_to = [Cause &
                     causeInvolvesSubstance.some(
                         substance_onto.hasStateOfAggregation.some(substance_onto.Liquid) &
                         substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                     isCauseOfDeviation.some(deviation_onto.HighPressure) &
                     causeInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity &
                                                       equipment onto.hasIntendedFunction.some(
                                                                                               process onto.Filling |
                                                                                               process onto.ModeIndependent))]
WaterHammer.comment = ["Synonyms: hydraulic shock, fluid hammer"]

class ThermalExpansion(Cause):
    equivalent_to = [Cause &
                     ((causeInvolvesSubstance.some(substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                                             substance_onto.Multiphase |
                                                                                             substance_onto.Gaseous) &
                                                   substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                       causeInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                                                                process onto.ModeIndependent)) &
                       causeRequiresBoundaryCondition.some(boundary_onto.ExternalFirePossible |
                                                           boundary_onto.LocatedOutside) &
                       isCauseOfDeviation.some(deviation_onto.HighPressure))
                      |
                      (causeInvolvesSubstance.some(substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                                             substance onto.Multiphase |
                                                                                             substance onto.Gaseous) &
                                                   substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                       causeInvolvesEquipmentEntity.some(equipment onto.hasApparatus.some(equipment onto.PressureVessel) &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.ModeIndependent)
                                                         ) &
                       causeRequiresBoundaryCondition.some(boundary_onto.ExternalFirePossible |
                                                           boundary onto.LocatedOutside) &
                       isCauseOfDeviation.some(deviation_onto.HighPressure)))]
```

Appendix Listing L-2: Class definitions of cause concepts Part II

```
class HighAmbientTemperature(Cause):
    equivalent to = [Cause &
                     (causeInvolvesEquipmentEntity.some(equipment onto.AirCooledCondenserEntity) &
                      isCauseOfDeviation.some(deviation_onto.HighTemperature |
                                              deviation_onto.HighPressure) &
                      causeRequiresBoundaryCondition.some(boundary_onto.LocatedOutside))]

class WrongRotatingSpeed(Cause):
    equivalent to = [Cause &
                     (
                     (isCauseOfDeviation.some(deviation onto.HighFlow |
                                              deviation onto.LowFlow) &
                      causeInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium |
                                                              substance_onto.Refrigerant)) &
                      causeInvolvesEquipmentEntity.some((equipment_onto.CompressorEntity |
                                                         equipment onto.PumpEntity) &
                                                        equipment onto.hasInstrumentation.some(equipment onto.FrequencyConverter) &
                                                        equipment onto.hasInstrumentation.some(equipment onto.SpeedController) &
                                                        equipment onto.hasInstrumentation.some(equipment onto.ElectricMotor)))
                     |
                     (isCauseOfDeviation.some(
                         deviation onto.HighPressure | deviation onto.HighTemperature) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.AirCooledCondenserEntity &
                                                        equipment_onto.hasInstrumentation.some(equipment_onto.ElectricMotor) &
                                                        equipment onto.hasInstrumentation.some(equipment onto.SpeedController) &
                                                        equipment onto.hasInstrumentation.some(equipment onto.FrequencyConverter))
                      ))]

class ConfusionOfSubstances(Cause):
    equivalent to = [Cause &
                     ((causeInvolvesEquipmentEntity.some(equipment_onto.SourceEntity &
                                                         equipment_onto.entityControlledBy.some(equipment_onto.Operator)) &
                       isCauseOfDeviation.some(deviation_onto.OtherThanComposition)))]

class WrongTankLinedUp(Cause):
    equivalent to = [Cause &
                     (causeInvolvesEquipmentEntity.some(equipment onto.TankTruckEntity &
                                                        equipment onto.hasIntendedFunction.some(
                                                            process_onto.Unloading)) &
                      isCauseOfDeviation.some(deviation_onto.OtherThanComposition))]

class OtherSubstanceFromUpstream(Cause):
    equivalent_to = [Cause &
                     ((causeRequiresBoundaryCondition.some(boundary_onto.UpstreamProcessInvolved) &
                       causeInvolvesSubstance.some(
                                                substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                       causeInvolvesEquipmentEntity.some(equipment onto.SourceEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                                                process_onto.ModeIndependent)) &
                       isCauseOfDeviation.some(deviation_onto.OtherThanComposition))
                     |
                      ((causeInvolvesSubstance.some(
                                                substance_onto.hasSpecificTask.some(substance_onto.InertGas)) &
                        causeInvolvesEquipmentEntity.some(equipment_onto.InertgasBlanketingEntity &
                                                          equipment onto.hasIntendedFunction.some(
                                                                                 process onto.Inerting)) &
                        isCauseOfDeviation.some(deviation onto.OtherThanComposition))))]

class ReducedDwellTime(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation_onto.OtherThanComposition) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.SettlingTankEntity &
                                                        equipment_onto.hasIntendedFunction.some(
                                                                                    process onto.Separating)))]

class ContaminationInUnloadingLines(Cause):
    equivalent to = [Cause &
                     (causeInvolvesEquipmentEntity.some((equipment_onto.hasPiping.some(
                                                         equipment_onto.TankTruckHose)) &
                                                        equipment_onto.hasIntendedFunction.some(
                                                                                  process_onto.Unloading)) &
                      causeRequiresBoundaryCondition.some(boundary onto.IntroductionOfImpurities |
                                                          boundary onto.IntroductionOfWater) &
                      isCauseOfDeviation.some(deviation onto.OtherThanComposition))]

class InadvertentContamination(Cause):
    equivalent to = [Cause &
                     ((causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfImpurities) &
                       causeInvolvesEquipmentEntity.some((equipment_onto.SourceEntity |
                                                          equipment onto.SettlingTankEntity |
                                                          equipment onto.PressureVesselEntity |
                                                          equipment onto.WetScrubberEntity |
                                                          equipment onto.PressureReceiverEntity) &
                                                          equipment onto.hasIntendedFunction.some(
                                                                    process onto.ModeIndependent)) &
                       isCauseOfDeviation.some(deviation onto.OtherThanComposition))
                     |
                     (causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfImpurities) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.TankTruckEntity &
                                                        equipment onto.hasIntendedFunction.some(
                                                                                  process onto.Unloading)) &
                      isCauseOfDeviation.some(deviation onto.OtherThanComposition)))]

class ContaminationByWaterAndTemperatureFallsBelowFreezingPoint(Cause):
    equivalent_to = [Cause &
                     (causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfWater) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.hasApparatus.some(
                                                              equipment_onto.AtmosphericStorageTank |
                                                              equipment_onto.PressureVessel) &
                                                        equipment onto.hasIntendedFunction.some(
                                                              process onto.Storing | process onto.ModeIndependent)) &
                      causeInvolvesSecondDeviation.some(deviation onto.OtherThanComposition) &
                      isCauseOfDeviation.some(deviation onto.LowTemperature))]

class ContaminationByWater(Cause):
    equivalent_to = [Cause &
                     ((causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfWater) &
                       causeInvolvesEquipmentEntity.some((equipment_onto.SettlingTankEntity |
                                                          equipment_onto.PressureVesselEntity |
                                                          equipment onto.WetScrubberEntity |
                                                          equipment onto.PressureReceiverEntity |
                                                          equipment onto.ShellTubeHeatExchangerEntity |
                                                          equipment onto.ShellTubeEvaporatorEntity |
                                                          equipment_onto.FinTubeEvaporatorEntity) &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.ModeIndependent |
                                                              process_onto.Filling)
                                                         ) &
                       isCauseOfDeviation.some(deviation onto.OtherThanComposition))
                      |
                      (causeRequiresBoundaryCondition.some(boundary onto.IntroductionOfWater) &
                       causeInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                         equipment_onto.hasIntendedFunction.some(
                                                                                    process_onto.Storing |
                                                                                    process_onto.Filling)) &
                       isCauseOfDeviation.some(deviation_onto.OtherThanComposition)))]
```

Appendix Listing L-3: Class definitions of cause concepts Part III

```
class MaterialDegradation(Cause):
    equivalent_to = [Cause &
                     (isCauseOfDeviation.some(deviation_onto.HighVibration) &
                      causeInvolvesEquipmentEntity.some(equipment_onto.hasApparatus.some(
                                                            equipment_onto.PressureVessel |
                                                            equipment_onto.AtmosphericStorageTank |
                                                            equipment_onto.Piping)))]

class ExternalLeakage(Cause):
    equivalent_to = [Cause &
                     ((causeInvolvesEquipmentEntity.some(equipment_onto.CompressorEntity) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Gaseous)) &  # liquid: because lubricant
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow))
                      |
                      (isCauseOfDeviation.some(deviation_onto.HighCorrosion) &
                       causeInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.ModeIndependent)))
                      |
                      (causeInvolvesEquipmentEntity.some((equipment_onto.PressureReceiverEntity |
                                                          equipment_onto.FinTubeEvaporatorEntity |
                                                          equipment_onto.ShellTubeEvaporatorEntity |
                                                          equipment_onto.AirCooledCondenserEntity |
                                                          equipment_onto.WetScrubberEntity) &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.ModeIndependent)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Multiphase)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow))
                      |
                      (causeInvolvesEquipmentEntity.some(equipment_onto.TankTruckEntity &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.Unloading)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow))
                      |
                      (causeInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.ModeIndependent)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Gaseous |
                                                                     substance_onto.Multiphase)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow))
                      |
                      (causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasInstrumentation.some(equipment_onto.BottomDrainValve |
                                                                equipment_onto.ThreeWayValve |
                                                                equipment_onto.ShutOffValve) &
                        equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Gaseous |
                                                                     substance_onto.Multiphase)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow))
                      |
                      (causeInvolvesEquipmentEntity.some(equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.ModeIndependent)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Gaseous |
                                                                     substance_onto.Multiphase) &
                           substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                       isCauseOfDeviation.some(deviation_onto.ElsewhereFlow)))]

class InternalLeakage(Cause):
    equivalent_to = [Cause &
                     ((isCauseOfDeviation.some(deviation_onto.ElsewhereFlow |
                                               deviation_onto.OtherThanComposition) &
                       causeInvolvesEquipmentEntity.some(equipment_onto.hasFixture.some(equipment_onto.HalfPipeCoilJacket |
                                                                                        equipment_onto.Jacket |
                                                                                        equipment_onto.PlatePackage) |
                                                         equipment_onto.hasPiping.some(equipment_onto.TubeCoil |
                                                                                       equipment_onto.TubeBundle)))
                      |
                      (isCauseOfDeviation.some(deviation_onto.ElsewhereFlow) &
                       causeInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.HeatingMedium)) &
                       causeInvolvesEquipmentEntity.some(equipment_onto.StabilizerColumnEntity |
                                                         equipment_onto.SteamDrivenReboilerEntity)))]

class ClosedOutletValve(Cause):
    equivalent_to = [Cause &
                     (
                      (causeInvolvesEquipmentEntity.some(equipment_onto.hasInstrumentation.some(equipment_onto.OutletValve) &
                                                         equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                                                         causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous)) &
                                                         isCauseOfDeviation.some(deviation_onto.HighPressure))
                       |
                       (causeInvolvesEquipmentEntity.some(equipment_onto.hasInstrumentation.some(equipment_onto.OutletValve) &
                                                          equipment_onto.hasApparatus.some(equipment_onto.Compressor)) &
                        causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous)) &
                        isCauseOfDeviation.some(deviation_onto.NoFlow | deviation_onto.HighPressure))))]
ClosedOutletValve.comment = ["Pump specific details are covered in pump ontology"]

class ClosedInletValve(Cause):
    equivalent_to = [Cause &
                     ((causeInvolvesEquipmentEntity.some(equipment_onto.hasInstrumentation.some(equipment_onto.InletValve) &
                                                         equipment_onto.hasApparatus.some(equipment_onto.PressureVessel |
                                                                                          equipment_onto.AtmosphericStorageTank)) &
                       causeInvolvesSubstance.some(substance_onto.hasStateOfAggregation.some(substance_onto.Liquid)) &
                       isCauseOfDeviation.some(deviation_onto.LowLevel | deviation_onto.NoFlow))
                      |
                      (causeInvolvesEquipmentEntity.some(equipment_onto.hasInstrumentation.some(equipment_onto.InletValve) &
                                                         equipment_onto.hasApparatus.some(equipment_onto.PressureVessel)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous)) &
                       isCauseOfDeviation.some(deviation_onto.LowPressure))
                      |
                      (causeInvolvesEquipmentEntity.some(equipment_onto.CompressorEntity &
                                                         equipment_onto.hasInstrumentation.some(equipment_onto.InletValve)) &
                       causeInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous)) &
                       isCauseOfDeviation.some(deviation_onto.NoFlow |
                                               deviation_onto.LowPressure))
                      |
                      (causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                         equipment_onto.hasInstrumentation.some(equipment_onto.InletValve) &
                                                         equipment_onto.hasIntendedFunction.some(process_onto.DeliverConstantVolumeFlow)) &
                       isCauseOfDeviation.some(deviation_onto.NoFlow)))]
ClosedInletValve.comment = ["Pump specific details are covered in pump ontology"]
```

Appendix Listing L-4: Class definitions of cause concepts Part VI

```
class LossOfCooling(Cause):
    equivalent to = [Cause &
                     (causeInvolvesEquipmentEntity.some(equipment onto.hasSubunit.some(equipment onto.CoolingSystem)) &
                      isCauseOfDeviation.some(deviation_onto.HighTemperature))]

class DeliveryOfHighVolatilityComponents(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes_onto.isCauseOfDeviation.some(deviation_onto.OtherThanComposition) &
                      causes onto.causeInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity) &
                      causes onto.causeInvolvesSubstance.some(substance onto.hasSpecificTask.some(substance onto.ProcessMedium)))]

class BlockedReboilerLines(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.isCauseOfDeviation.some(deviation_onto.NoFlow) &
                      causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity) &
                      causes_onto.causeInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.ProcessMedium)))]

class MalfunctionLubricationSystem(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.Lubricant)) &
                      causes onto.causeInvolvesEquipmentEntity.some(equipment onto.CompressorEntity &
                                                                    equipment_onto.hasSubunit.some(equipment_onto.LubricationSystem)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighTemperature))]

MalfunctionLubricationSystem.comment = ["Issues described in 'How to limit fire and explosion hazards with oil-flooded rotary screw compressors' by
Steven J. Luzik"]

class InsufficientVentilation(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.CompressorEntity) &
                      causes_onto.causeInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium |
                                                              substance_onto.Refrigerant)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.HighTemperature))]

class NoSteamFlow(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.isCauseOfDeviation.some(deviation_onto.NoFlow) &
                      causes_onto.causeInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.HeatingMedium)) &
                      causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity))]

class LessSteamFlow(causes_onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.isCauseOfDeviation.some(deviation onto.LowFlow) &
                      causes onto.causeInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.HeatingMedium)) &
                      causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity))]

class MoreSteamFlow(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes_onto.isCauseOfDeviation.some(deviation_onto.HighFlow) &
                      causes_onto.causeInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.HeatingMedium)) &
                      causes onto.causeInvolvesEquipmentEntity.some(equipment onto.SteamDrivenReboilerEntity))]

class NonCondensables(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.AirCooledCondenserEntity) &
                      causes_onto.causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfAir) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.OtherThanComposition))]

NonCondensables.comment = ["Eigentlich wird hier noch high pressure benötigt"]

class Fouling(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesEquipmentEntity.some(equipment onto.AirCooledCondenserEntity) &
                       causes_onto.causeRequiresBoundaryCondition.some(boundary_onto.IntroductionOfImpurities) &
                       causes_onto.isCauseOfDeviation.some(deviation_onto.HighTemperature))
                     |
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.ShellTubeHeatExchangerEntity |
                                                                    equipment onto.ShellTubeEvaporatorEntity) &
                      causes onto.causeInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.Refrigerant)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.HighTemperature |
                                                          deviation onto.OtherThanComposition)))]

class IncorrectSetPointControlValve(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     ((causes onto.causeInvolvesEquipmentEntity.some(
                         equipment onto.hasInstrumentation.some(equipment onto.FlowControlValve) &
                         equipment onto.hasIntendedFunction.some(process onto.FlowControl)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.HighFlow |
                                                           deviation onto.NoFlow |
                                                           deviation onto.LowFlow))
                      |
                      (causes_onto.causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasInstrumentation.some(equipment_onto.PressureControlValve) &
                         equipment_onto.hasIntendedFunction.some(process_onto.PressureControl)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.HighPressure |
                                                           deviation onto.LowPressure)))]

class BypassOpened(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasSubunit.some(equipment_onto.Bypass) &
                         equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)
                     ) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighFlow))
                     ]

class ValveIntactUnintentionallyClosed(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes_onto.causeInvolvesEquipmentEntity.some((equipment_onto.ValveEntity |
                                                                      equipment_onto.InertgasBlanketingEntity) &
                                                                     equipment_onto.hasInstrumentation.some(
                                                                         equipment_onto.FlowControlValve |
                                                                         equipment_onto.PressureControlValve |
                                                                         equipment_onto.ShutOffValve |
                                                                         equipment onto.ThreeWayValve) &
                                                                     equipment onto.hasIntendedFunction.some(
                                                                         process onto.ModeIndependent)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.NoFlow))
                       |
                      (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                     (equipment_onto.hasInstrumentation.some(equipment_onto.OutletValve) |
                                                                      equipment_onto.hasInstrumentation.some(equipment_onto.InletValve)) &
                                                                     causes_onto.isCauseOfDeviation.some(
                                                                         deviation onto.NoFlow |
                                                                         deviation onto.HighTemperature))
                      ))]
```

Appendix Listing L-5: Class definitions of cause concepts Part V

```
class ValveStuckOpen(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesEquipmentEntity.some(equipment onto.ValveEntity &
                                                                     equipment_onto.hasInstrumentation.some(
                                                                         equipment_onto.FlowControlValve |
                                                                         equipment_onto.PressureControlValve) &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process_onto.FlowControl |
                                                                         process onto.PressureControl)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.HighFlow))
                     |
                     (causes onto.causeInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowLevel)))]

class ValvePartiallyOpened(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(equipment onto.ValveEntity &
                                                                    equipment onto.hasInstrumentation.some(
                                                                        equipment onto.ShutOffValve)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowFlow))]

class PluggedRestrictionOrifice(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.Orifice) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.NoFlow))]

class ValveWronglyClosed(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(
                                    equipment onto.hasInstrumentation.some(equipment onto.FlowControlValve) &
                                    equipment onto.hasIntendedFunction.some(process onto.FlowControl)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.NoFlow))]

class ValveWronglyOpened(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(
                                    equipment onto.hasInstrumentation.some(equipment onto.FlowControlValve) &
                                    equipment onto.hasIntendedFunction.some(process onto.FlowControl |
                                                                            process onto.ModeIndependent)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighFlow))]

class PumpingAgainstPolymerizedLine(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.isCauseOfDeviation.some(deviation onto.NoFlow) &
                      causes onto.causeInvolvesSubstance.some(
                          substance onto.hasStabilityReactivityInformation.some(
                              substance onto.PolymerizesExothermicallyWhenExposedToLight |
                              substance_onto.PolymerizesExothermicallyWhenExposedToHeat |
                              substance_onto.PolymerizesExothermicallyWithoutInhibitor)) &
                      causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                    equipment onto.hasIntendedFunction.some(
                                                                        process onto.DeliverConstantVolumeFlow)))]

class ImpellerFault(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                    equipment_onto.hasMaterialTransferEquipment.some(
                                                                        equipment_onto.CentrifugalPump) &
                                                                    equipment_onto.hasIntendedFunction.some(
                                                                        process_onto.DeliverConstantVolumeFlow)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowFlow))]

class WrongImpeller(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                    equipment_onto.hasMaterialTransferEquipment.some(
                                                                        equipment_onto.CentrifugalPump) &
                                                                    equipment_onto.hasIntendedFunction.some(
                                                                        process_onto.DeliverConstantVolumeFlow)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighFlow))]

class MissingImpeller(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                    equipment_onto.hasMaterialTransferEquipment.some(
                                                                        equipment_onto.CentrifugalPump) &
                                                                    equipment_onto.hasFixture.some(
                                                                        equipment_onto.Impeller) &
                                                                    equipment_onto.hasIntendedFunction.some(
                                                                        process_onto.DeliverConstantVolumeFlow)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.NoFlow))]

class PumpOperationFailure(causes onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     ((causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                     equipment_onto.hasMaterialTransferEquipment.some(
                                                                 equipment_onto.CentrifugalPump) &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process onto.DeliverConstantVolumeFlow)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.ReverseFlow)))]

class PumpIncorrectlySet(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                   equipment_onto.hasIntendedFunction.some(
                                                                       process_onto.DeliverConstantVolumeFlow)) &
                     causes onto.isCauseOfDeviation.some(deviation onto.HighFlow)]

class OperationBelowMinimumFlowRate(causes onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                    equipment_onto.hasMaterialTransferEquipment.some(
                                                                equipment_onto.CentrifugalPump) &
                                                                    equipment_onto.hasIntendedFunction.some(
                                                                        process onto.DeliverConstantVolumeFlow)
                                                                    ) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowFlow))]

class EntrainedAir(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                    equipment_onto.hasIntendedFunction.some(
                                                                        process onto.DeliverConstantVolumeFlow)) &
                      causes onto.causeRequiresBoundaryCondition.some(boundary onto.IntroductionOfAir) &
                      causes onto.isCauseOfDeviation.some(deviation onto.OtherThanComposition))]
```

Appendix Listing L-6: Class definitions of cause concepts Part VI

```
class DeadHeadingOfPump(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                     equipment_onto.hasInstrumentation.some(
                                                                         equipment_onto.OutletValve) &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process_onto.DeliverConstantVolumeFlow |
                                                                         equipment_onto.NormalOperation)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.NoFlow)))]
DeadHeadingOfPump.comment = ["Occurs when the pump's discharge is closed (blockage or closed valve)"]

class InsufficientNPSH(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                                    equipment_onto.hasIntendedFunction.some(
                                                                        process_onto.DeliverConstantVolumeFlow)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowPressure))]

class PumpSealFailure(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                     equipment onto.hasIntendedFunction.some(
                                                                         process onto.DeliverConstantVolumeFlow)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.ElsewhereFlow)))]

class ChargingFailure(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.isCauseOfDeviation.some(deviation onto.OtherThanComposition |
                                                          deviation onto.OtherSequence) &
                      causes onto.causeInvolvesEquipmentEntity.some(equipment onto.ReactorEntity))]

class DosingFailure(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     (causes_onto.isCauseOfDeviation.some(deviation_onto.OtherThanComposition |
                                                          deviation_onto.OtherSequence) &
                      causes onto.causeInvolvesEquipmentEntity.some(equipment onto.ReactorEntity))]

class AbnormalHeatInput(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes_onto.causeInvolvesEquipmentEntity.some((equipment_onto.PressureVesselEntity |
                                                                      equipment_onto.PlateHeatExchangerEntity |
                                                                      equipment_onto.AirCooledCondenserEntity |
                                                                      equipment_onto.ShellTubeHeatExchangerEntity |
                                                                      equipment_onto.ConnectionPipeEntity) &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process onto.ModeIndependent |
                                                                         process onto.Storing)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.HighTemperature))
                     |
                      (causes_onto.causeInvolvesEquipmentEntity.some(
                          equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                          equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                       causes_onto.isCauseOfDeviation.some(deviation_onto.HighTemperature))
                      |
                      (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                                     equipment onto.hasIntendedFunction.some(
                                                                         process onto.Filling |
                                                                         process onto.Storing)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.HighTemperature))
                      |
                      (causes_onto.causeInvolvesEquipmentEntity.some(
                          equipment_onto.hasSubunit.some(equipment_onto.HeatingSystem)) &
                       causes_onto.isCauseOfDeviation.some(deviation_onto.HighTemperature)))]

class LeakingDrainValve(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(
                         equipment onto.hasInstrumentation.some(equipment onto.BottomDrainValve) &
                         equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                      causes_onto.causeInvolvesSubstance.some(
                          substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                    substance_onto.Multiphase)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.ElsewhereFlow))]

class AbruptReliefOfContent(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesSubstance.some(
                         substance onto.hasStateOfAggregation.some(substance onto.Gaseous)) &
                      (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity |
                                                                     equipment_onto.hasApparatus.some(
                                                                         equipment onto.PressureVessel))) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowTemperature))]
AbruptReliefOfContent.comment = ["Abrupt relief can lead to ignition of flammable mixture"]

class CoolingFailure(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasSubunit.some(equipment_onto.CoolingSystem)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighTemperature))]

class BlockedInflowLine(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesSubstance.some(
                         substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                   substance onto.Multiphase)) &
                       causes_onto.isCauseOfDeviation.some(deviation_onto.LowLevel) &
                       causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process_onto.Filling)))
                      |
                      (causes onto.causeInvolvesSubstance.some(
                          substance onto.hasStateOfAggregation.some(substance onto.Gaseous)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.LowPressure) &
                       causes onto.causeInvolvesEquipmentEntity.some(
                           equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                           equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)
                      ))
                      |
                      (causes_onto.causeInvolvesSubstance.some(
                          substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                    substance_onto.Multiphase)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.LowLevel) &
                       causes onto.causeInvolvesEquipmentEntity.some(
                           equipment onto.hasApparatus.some(equipment onto.PressureVessel) &
                           equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)))
                      |
                      (causes_onto.causeInvolvesEquipmentEntity.some(equipment_onto.WetScrubberEntity &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process_onto.ModeIndependent)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.NoFlow | deviation onto.LowFlow |
                                                           deviation onto.LowLevel | deviation onto.LowPressure))
                      |
                      (causes onto.causeInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity &
                                                                     equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                       causes_onto.causeInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                       causes_onto.isCauseOfDeviation.some(deviation_onto.NoFlow)))]
```

Appendix Listing L-7: Class definitions of cause concepts Part VII

```
class ExcessiveInflow(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesSubstance.some(
                         substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous |
                                                                   substance_onto.Multiphase |
                                                                   substance_onto.Liquid)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighFlow) &
                      causes_onto.causeInvolvesEquipmentEntity.some(
                          equipment onto.hasApparatus.some(equipment onto.PressureVessel) &
                          equipment onto.hasIntendedFunction.some(process onto.ModeIndependent)
                      ))
                     |
                     (causes onto.causeInvolvesSubstance.some(
                         substance_onto.hasStateOfAggregation.some(
                             substance_onto.Liquid | substance_onto.Multiphase)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighLevel) &
                      causes onto.causeInvolvesEquipmentEntity.some(
                          equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank |
                                                           equipment onto.PressureVessel) &
                          equipment onto.hasIntendedFunction.some(process onto.Filling))
                     )
                     |
                     (causes onto.causeInvolvesSubstance.some(
                         substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighFlow) &
                      causes onto.causeInvolvesEquipmentEntity.some(equipment onto.InertgasBlanketingEntity &
                                                                    equipment onto.hasIntendedFunction.some(
                                                                       process onto.Inerting)
                                                                    )
                     )
                     )
                    ]

class LossOfInflow(causes_onto.Cause):
    equivalent_to = [causes_onto.Cause &
                     ((causes onto.causeInvolvesSubstance.some(
                         substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                   substance onto.Multiphase)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowLevel) &
                      causes_onto.causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank) &
                         equipment_onto.hasIntendedFunction.some(process_onto.Filling)
                      ))
                     |
                      (causes_onto.causeInvolvesSubstance.some(
                          substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                    substance onto.Multiphase)) &
                       causes onto.isCauseOfDeviation.some(deviation onto.LowLevel) &
                       causes onto.causeInvolvesEquipmentEntity.some(
                           equipment_onto.hasApparatus.some(equipment_onto.PressureVessel)  &
                           equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)
                       ))
                     )
                    ]

class IncorrectIndicationOfFillingLevel(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     ((causes onto.causeInvolvesSubstance.some(
                         substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                   substance_onto.Multiphase)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighLevel) &
                      causes_onto.causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank) &
                         equipment_onto.hasIntendedFunction.some(process_onto.Filling)))
                     |
                     (causes onto.causeInvolvesSubstance.some(
                         substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                   substance onto.Multiphase)) &
                      causes_onto.isCauseOfDeviation.some(deviation_onto.HighLevel) &
                      causes_onto.causeInvolvesEquipmentEntity.some(
                          equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                          equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)))
                     )
                    ]

class IncorrectFilling(causes onto.Cause):
    equivalent to = [causes onto.Cause &
                     (causes onto.causeInvolvesEquipmentEntity.some(
                         equipment_onto.hasApparatus.some(equipment_onto.PressureVessel |
                                                          equipment_onto.AtmosphericStorageTank) &
                         equipment onto.entityControlledBy.some(equipment onto.Operator) &
                         equipment onto.hasIntendedFunction.some(process onto.Filling)) &
                      causes onto.isCauseOfDeviation.some(deviation onto.LowLevel |
                                                          deviation onto.HighLevel |
                                                          deviation onto.LowFlow |
                                                          deviation onto.HighFlow))]
```

Appendix Listing L-8: Class definitions of cause concepts Part VIII

Appendix M Formal ontology class definitions effects

```
class BackContaminationOfSupply(Effect):
    equivalent to = [Effect &
                     ((effectInvolvesEquipmentEntity.some(equipment onto.PumpEntity) &
                       isEffectOfDeviation.some(deviation onto.ReverseFlow)))]

class InsufficientFilling(Effect):
    equivalent to = [Effect & (
             (effectImpliedByCause.some(causes onto.BlockedInflowLine | causes onto.LossOfInflow) &
              isEffectOfDeviation.some(deviation onto.NoFlow | deviation onto.LowLevel) &
              effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                 equipment onto.hasIntendedFunction.some(process onto.Filling)))
             |
             (isEffectOfDeviation.some(deviation_onto.NoFlow) &
              effectOfPropagatedCause.value(True) &
              effectInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                 equipment_onto.hasIntendedFunction.some(process_onto.Filling)))]

class UnintendedExothermicPolymerization(Effect):
    equivalent to = [Effect &
                     (
                             (effectInvolvesSubstance.some(
                                 substance_onto.hasStabilityReactivityInformation.some(
                                     substance_onto.PolymerizesExothermicallyWithoutInhibitor) &
                                 effectImpliedByCause.some(causes_onto.TooLittleStabilizer |
                                                           causes_onto.TooLittleInhibitor |
                                                           causes_onto.InadvertentContamination) &
                                 effectInvolvesEquipmentEntity.some((equipment onto.ConnectionPipeEntity |
                                                                     equipment onto.PressureVesselEntity |
                                                                     equipment onto.WetScrubberEntity |
                                                                     equipment onto.StorageTankEntity |
                                                                     equipment_onto.SettlingTankEntity) &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process_onto.ModeIndependent)
                                                                     )))
                             |
                             (effectInvolvesSubstance.some(
                                 substance onto.hasStabilityReactivityInformation.some(
                                     substance onto.PolymerizesExothermicallyWhenExposedToHeat)) &
                              effectImpliedByCause.some(causes onto.AbnormalHeatInput) &
                              effectImpliedByUnderlyingcause.some(causes onto.ExternalFire) &
                              effectInvolvesEquipmentEntity.some((equipment_onto.ConnectionPipeEntity |
                                                                  equipment_onto.PressureVesselEntity |
                                                                  equipment_onto.StorageTankEntity |
                                                                  equipment_onto.SettlingTankEntity) &
                                                                  equipment onto.hasIntendedFunction.some(
                                                                      process onto.ModeIndependent))
                              )
                             |
                             (effectInvolvesSubstance.some(
                                 substance_onto.hasStabilityReactivityInformation.some(
                                     substance_onto.PolymerizesExothermicallyWithoutInhibitor)) &
                              isEffectOfDeviation.some(deviation_onto.HighTemperature) &
                              effectInvolvesEquipmentEntity.some(
                                  equipment onto.hasIntendedFunction.some(process onto.ModeIndependent)) &
                              effectImpliedByCause.some(causes onto.TooLittleInhibitor)
                              )
                     )]

class PotentialViolentReactionWithOxidizers(Effect):
    equivalent_to = [Effect &
                     (effectInvolvesEquipmentEntity.some(
                         equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank) &
                                                          equipment onto.hasIntendedFunction.some(
                                                                  process onto.Filling)) &
                      effectInvolvesSubstance.some(
                          substance onto.hasStabilityReactivityInformation.some(
                              substance onto.ReactsViolentlyWithOxidizer |
                              substance onto.IncompatibleToStrongOxidizers)) &
                      isEffectOfDeviation.some(deviation_onto.OtherThanComposition) &
                      effectImpliedByCause.some(causes_onto.WrongTankLinedUp |
                                                causes_onto.ConfusionOfSubstances))]

class AbnormalEvaporation(Effect):
    equivalent to = [Effect &
                     ((effectInvolvesSubstance.some(
                        substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                  substance_onto.Multiphase) &
                        substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                       effectInvolvesEquipmentEntity.some(
                          equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                          equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)
                       ) &
                       effectImpliedByUnderlyingcause.some(causes onto.ExternalFire) &
                       isEffectOfDeviation.some(deviation onto.HighTemperature))
                      |
                      (effectInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.Filling |
                                                              process_onto.Storing)) &
                       effectInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance onto.Multiphase)) &
                       effectImpliedByCause.some(causes onto.AbnormalHeatInput))
                      |
                      (effectInvolvesEquipmentEntity.some(equipment onto.PressureVessel &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.Filling |
                                                              process_onto.Storing)) &
                       effectInvolvesSubstance.some(substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                                              substance_onto.Multiphase)) &
                       effectImpliedByCause.some(causes onto.AbnormalHeatInput))
                      |
                      (isEffectOfDeviation.some(deviation onto.HighTemperature) &
                       effectInvolvesSubstance.some(substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                       effectInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity) &
                       effectImpliedByCause.some(causes_onto.MoreSteamFlow))
                      |
                      (effectImpliedByCause.some(causes_onto.DeliveryOfHighVolatilityComponents) &
                       effectInvolvesSubstance.some(
                           substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                       effectInvolvesEquipmentEntity.some(equipment onto.SteamDrivenReboilerEntity))
                     )]
```

Appendix Listing M-1: Class definitions of effect concepts Part I

```
class AccumulationOfImpurities(Effect):
    equivalent_to = [Effect &
                     (effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.Filling)) &
                      effectImpliedByCause.some(causes_onto.ContaminationInUnloadingLines |
                                                causes_onto.InadvertentContamination) &
                      effectInvolvesSubstance.some(
                          substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                    substance onto.Multiphase)) &
                      effectOfPropagatedCause.value(True) &
                      isEffectOfDeviation.some(deviation onto.OtherThanComposition))]

class BacteriaGrowth(Effect):
    equivalent_to = [Effect &
                     ((effectInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                          equipment onto.hasIntendedFunction.some(
                                                              process onto.Storing)) &
                       effectInvolvesSubstance.some(
                           substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                     substance onto.Multiphase)) &
                       effectImpliedByCause.some(causes onto.ContaminationByWater) &
                       isEffectOfDeviation.some(deviation onto.OtherThanComposition)
                       )
                      |
                      (effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                          equipment onto.hasIntendedFunction.some(
                                                              process onto.Storing)) &
                       effectImpliedByCause.some(causes onto.ContaminationInUnloadingLines |
                                                 causes onto.InadvertentContamination) &
                       effectInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Multiphase)) &
                       effectOfPropagatedCause.value(True) &
                       isEffectOfDeviation.some(deviation_onto.OtherThanComposition)))]

class GenerationOfElectrostaticCharge(Effect):
    equivalent_to = [Effect &
                     (effectInvolvesEquipmentEntity.some((equipment onto.TankTruckEntity |
                                                          equipment_onto.ConnectionPipeEntity) &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process_onto.Unloading |
                                                             process_onto.ModeIndependent)) &
                      effectInvolvesSubstance.some(
                          substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                    substance onto.Multiphase)) &
                      isEffectOfDeviation.some(deviation onto.HighFlow))]

class Overfilling(Effect):
    equivalent_to = [Effect &
                     ((effectInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.Filling)) &
                       effectInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance onto.Multiphase)) &
                       isEffectOfDeviation.some(deviation onto.HighLevel))
                      |
                      (effectInvolvesEquipmentEntity.some(
                          equipment_onto.hasApparatus.some(equipment_onto.PressureVessel) &
                          equipment_onto.hasPiping.some(equipment_onto.VentPipe) &
                          equipment_onto.hasIntendedFunction.some(process_onto.Filling |
                                                                  process_onto.ModeIndependent)) &
                       effectInvolvesSubstance.some(
                           substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                     substance onto.Multiphase)) &
                       isEffectOfDeviation.some(deviation onto.HighLevel))
                      |
                      (isEffectOfDeviation.some(deviation_onto.HighFlow) &
                       effectInvolvesEquipmentEntity.some(
                           equipment_onto.hasApparatus.some(equipment_onto.PressureVessel |
                                                            equipment_onto.AtmosphericStorageTank |
                                                            equipment onto.OpenVessel) &
                           equipment onto.hasIntendedFunction.some(process onto.Filling)) &
                       effectImpliedByCause.some(causes onto.BypassOpened |
                                                 causes onto.IncorrectSetPointControlValve |
                                                 causes onto.WrongImpeller |
                                                 causes onto.IncorrectFilling) &
                       effectInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance onto.Multiphase)))
                      |
                      (effectInvolvesEquipmentEntity.some(
                          equipment onto.hasApparatus.some(equipment onto.PressureVessel |
                                                           equipment onto.AtmosphericStorageTank)) &
                       effectImpliedByCause.some(causes onto.IncorrectIndicationOfFillingLevel) &
                       effectInvolvesSubstance.some(
                           substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                     substance_onto.Multiphase))
                       ))]

class PressureExceedingDesignPressure(Effect):
    equivalent_to = [Effect &
                     ((isEffectOfDeviation.some(deviation onto.HighPressure) &
                          effectInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity |
                                                             equipment_onto.hasApparatus.some(equipment_onto.PressureVessel)))
                        |
                        (effectInvolvesEquipmentEntity.some((equipment_onto.AirCooledCondenserEntity |
                                                             equipment_onto.ShellTubeEvaporatorEntity |
                                                             equipment_onto.SteamDrivenReboilerEntity |
                                                             equipment onto.FinTubeEvaporatorEntity |
                                                             equipment onto.CompressorEntity) &
                                                             equipment onto.hasIntendedFunction.some(process onto.ModeIndependent)) &
                         effectImpliedByCause.some(causes onto.BlockedOutflowLine))
                        |
                        (effectOfPropagatedCause.value(True) &
                         isEffectOfDeviation.some(deviation_onto.HighPressure) &
                         effectInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                            equipment_onto.hasIntendedFunction.some(
                                                                process_onto.Filling | process_onto.Storing | process_onto.ModeIndependent)))
                        |
                        (isEffectOfDeviation.some(deviation onto.HighPressure) &
                         effectImpliedByCause.some(causes onto.ThermalExpansion | causes onto.InsufficientThermalOutbreathing) &
                         effectInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity |
                                                            equipment_onto.hasApparatus.some(
                                                                equipment_onto.PressureVessel |
                                                                equipment_onto.AtmosphericStorageTank)))
                        |
                        (effectInvolvesEquipmentEntity.some(
                            equipment onto.hasMaterialTransferEquipment.some(equipment onto.ReciprocatingPump)) &
                         effectImpliedByCause.some(causes onto.DeadHeadingOfPump |
                                                   causes onto.PumpingAgainstPolymerizedLine))
                        |
                        (effectInvolvesEquipmentEntity.some(
                            equipment_onto.hasIntendedFunction.some(process_onto.FlowControl | process_onto.ModeIndependent)) &
                         effectImpliedByCause.some(causes_onto.ValveClosedPressureBuildUpInPiping))
                        |
                        PotentialViolentReactionWithOxidizers)]
```

Appendix Listing M-2: Class definitions of effect concepts Part II

```
class Fracture(Effect):
    equivalent_to = [Effect &
                     (
                      (effectImpliedByCause.some(causes_onto.PhysicalImpact) &
                       isEffectOfDeviation.some(deviation_onto.ElsewhereFlow) &
                       effectInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.ModeIndependent)))
                      |
                      PressureExceedingDesignPressure)]

class FatigueFracture(Effect):
    equivalent_to = [Effect &
                     ((isEffectOfDeviation.some(deviation_onto.HighVibration) &
                       effectImpliedByCause.some(causes_onto.MaterialDegradation))
                      |
                      (isEffectOfDeviation.some(deviation onto.HighVibration) &
                       effectInvolvesEquipmentEntity.some(equipment onto.PumpEntity))
                      |
                      (isEffectOfDeviation.some(deviation onto.HighVibration) &
                       effectInvolvesEquipmentEntity.some(equipment onto.ConnectionPipeEntity)))]

class BrittleFracture(Effect):
    equivalent_to = [Effect &
                     isEffectOfDeviation.some(deviation_onto.LowTemperature) &
                     effectImpliedByCause.some(causes onto.MaterialDegradation)]

BrittleFracture.comment = ["Source: Managing Cold Temperature and Brittle Fracture Hazards in Pressure "
                           "Vessels by Daniel J. Benac, Nicholas Cherolis & David Wood",
                           "Requires crack in high stress region",
                           "sudden and unexpected failure",
                           "https://www.psenterprise.com/sectors/oil-and-gas/brittle-fracture"]

class DrainlineFracture(Effect):
    equivalent_to = [Effect &
                     ((isEffectOfDeviation.some(deviation onto.OtherThanComposition) &
                       effectInvolvesSubstance.some(
                           substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                     substance onto.Multiphase)) &
                       effectInvolvesEquipmentEntity.some(
                           equipment_onto.hasInstrumentation.some(equipment_onto.BottomDrainValve) &
                           equipment_onto.hasIntendedFunction.some(process_onto.ModeIndependent)) &
                       effectRequiresBoundaryCondition.some(
                           boundary_onto.AmbientTemperatureCanDropBelowFreezingPoint))
                      |
                      (isEffectOfDeviation.some(deviation onto.LowTemperature) &
                       effectInvolvesSecondDeviation.some(deviation onto.OtherThanComposition) &
                       effectInvolvesSubstance.some(
                           substance onto.hasStateOfAggregation.some(substance onto.Multiphase |
                                                                     substance_onto.Liquid)) &
                       effectInvolvesEquipmentEntity.some(
                           equipment_onto.hasInstrumentation.some(equipment_onto.BottomDrainValve) &
                           equipment_onto.hasIntendedFunction.some(process_onto.Storing | process_onto.ModeIndependent)))
                      |
                      (effectImpliedByCause.some(causes_onto.FreezeUp))
                      |
                      (isEffectOfDeviation.some(deviation onto.LowTemperature) &
                       effectInvolvesSubstance.some(
                           substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                     substance_onto.Multiphase)) &
                       effectInvolvesEquipmentEntity.some(
                           equipment_onto.hasInstrumentation.some(equipment_onto.BottomDrainValve)) &
                       effectRequiresBoundaryCondition.some(boundary_onto.IntroductionOfWater) &
                       effectInvolvesSiteInformation.some(
                           site information.hasMinimumAmbientTemperatureInKelvin <= 273.15)))]

DrainlineFracture.comment = ["second part of definition accomplishes for double jeopardy, T_low and X_other"]

class InsufficientInertization(Effect):
    equivalent_to = [Effect &
                     (effectInvolvesEquipmentEntity.some(equipment_onto.InertgasBlanketingEntity) &
                      isEffectOfDeviation.some(deviation_onto.NoFlow |
                                               deviation onto.OtherThanComposition) &
                      effectInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.InertGas)))
                     |
                     (isEffectOfDeviation.some(deviation onto.OtherThanComposition |
                                               deviation onto.NoFlow) &
                      effectInvolvesEquipmentEntity.some(
                          equipment_onto.hasPiping.some(equipment_onto.BlanketingGasline) &
                          equipment onto.hasIntendedFunction.some(process onto.Inerting)) &
                      effectImpliedByCause.some(causes onto.NoInertgasSupply |
                                                causes onto.OtherSubstanceFromUpstream))]

class PoorSeparation(Effect):
    equivalent_to = [Effect &
                     ((isEffectOfDeviation.some(deviation onto.OtherThanComposition) &
                               effectImpliedByCause.some(causes_onto.ConfusionOfSubstances |
                                                         causes_onto.OtherSubstanceFromUpstream) &
                               effectInvolvesSubstance.some(
                                   substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                               effectInvolvesEquipmentEntity.some(equipment onto.SettlingTankEntity &
                                                                  equipment onto.hasIntendedFunction.some(process onto.Separating)))
                              |
                              (isEffectOfDeviation.some(deviation onto.OtherThanComposition) &
                               effectInvolvesSubstance.some(
                                   substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                               effectImpliedByCause.some(causes_onto.ReducedDwellTime) &
                               effectInvolvesEquipmentEntity.some(equipment_onto.SettlingTankEntity &
                                                                  equipment_onto.hasIntendedFunction.some(process_onto.Separating)))
                              |
                              (effectInvolvesSubstance.some(
                                   substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                               effectImpliedByCause.some(causes onto.BypassOpened |
                                                         causes_onto.ValveWronglyOpened |
                                                         causes_onto.IncorrectSetPointControlValve) &
                               isEffectOfDeviation.some(deviation_onto.HighFlow) &
                               effectInvolvesEquipmentEntity.some(equipment_onto.SettlingTankEntity &
                                                                  equipment_onto.hasIntendedFunction.some(
                                                                      process_onto.Separating | process_onto.ModeIndependent)))
                     )]

class PoolFormation(Effect):
    equivalent_to = [Effect &
                     ((effectInvolvesSubstance.some(
                       substance_onto.hasStateOfAggregation.some(substance_onto.Liquid |
                                                                 substance_onto.Multiphase)) &
                       (effectImpliedByCause.some(causes_onto.ExternalLeakage |
                                                  causes_onto.PumpSealFailure |
                                                  causes onto.DrainValveInadvertentlyOpened |
                                                  causes onto.HoseIncorrectlyConnected)))
                      |
                      (isEffectOfDeviation.some(deviation onto.HighCorrosion) &
                       effectOfPropagatedCause.value(True)))
                     ]
```

Appendix Listing M-3: Class definitions of effect concepts Part III

```
class EmptyingOfContainer(Effect):
    equivalent_to = [Effect &
                     ((effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.Emptying)) &
                      isEffectOfDeviation.some(deviation_onto.LowLevel | deviation_onto.NoFlow | deviation_onto.LowFlow) &
                      effectImpliedByCause.some(causes_onto.LossOfInflow |
                                                causes_onto.IncorrectFilling |
                                                causes onto.ClosedInletValve))
                     |
                     (effectInvolvesEquipmentEntity.some(equipment onto.SettlingTankEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                             process onto.Separating)) &
                      isEffectOfDeviation.some(deviation_onto.LowLevel | deviation_onto.NoFlow | deviation_onto.LowFlow) &
                      effectImpliedByCause.some(causes_onto.LossOfInflow | causes_onto.ValveWronglyClosed |
                                                causes_onto.IncorrectFilling | causes_onto.ClosedInletValve))
                     |
                     (effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                             process onto.Emptying)) &
                      effectInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                      isEffectOfDeviation.some(deviation onto.NoFlow) &
                      effectOfPropagatedCause.value(True))
                     |
                     (effectInvolvesEquipmentEntity.some(equipment onto.SettlingTankEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                             process onto.ModeIndependent)) &
                      effectInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                      isEffectOfDeviation.some(deviation onto.NoFlow) &
                      effectOfPropagatedCause.value(True)))]

class LossOfMechanicalIntegrity(Effect):
    equivalent_to = [Effect &
                     ((effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity |
                                                          equipment onto.SettlingTankEntity |
                                                          equipment onto.WetScrubberEntity |
                                                          equipment onto.PressureVesselEntity |
                                                          equipment_onto.PressureReceiverEntity |
                                                          equipment_onto.ReactorEntity) &
                      effectImpliedByCause.some(
                          causes_onto.InsufficientThermalInbreathing |
                          causes_onto.DrainValveInadvertentlyOpened))
                     |
                     (effectImpliedByCause.some(causes onto.MechanicalFailureOfSupport) &
                      isEffectOfDeviation.some(deviation onto.ElsewhereFlow) &
                      effectInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity &
                                                         equipment onto.hasIntendedFunction.some(
                                                             process_onto.ModeIndependent)))
                     |
                     ((effectInvolvesEquipmentEntity.some((equipment_onto.StorageTankEntity) &
                                                          equipment_onto.hasIntendedFunction.some(
                                                              process_onto.Emptying |
                                                              process_onto.Storing))) &
                      isEffectOfDeviation.some(deviation onto.LowPressure))
                     |
                     ((effectInvolvesEquipmentEntity.some(equipment onto.SettlingTankEntity &
                                                          equipment onto.hasIntendedFunction.some(
                                                              process_onto.Separating |
                                                              process_onto.ModeIndependent))) &
                      isEffectOfDeviation.some(deviation_onto.LowPressure)))]

LossOfMechanicalIntegrity.comment = ["Underpressure", "Armospheric Tank Failures: Mechanisms and an Unexpected Case Study"]

class GasDispersion(Effect):
    equivalent_to = [Effect &
                     (effectInvolvesSubstance.some(
                         substance_onto.hasStateOfAggregation.some(substance_onto.Gaseous)) &
                      effectImpliedByCause.some(causes_onto.ExternalLeakage))]

class FluidCirculatesInsidePump(Effect):
    equivalent_to = [Effect &
                     ((isEffectOfDeviation.some(deviation onto.NoFlow) &
                       effectInvolvesEquipmentEntity.some(
                          equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump) &
                          equipment_onto.hasIntendedFunction.some(process_onto.DeliverConstantVolumeFlow)) &
                      effectImpliedByCause.some(causes_onto.PumpingAgainstPolymerizedLine |
                                                causes_onto.DeadHeadingOfPump))
                     |
                     (isEffectOfDeviation.some(deviation onto.NoFlow) &
                      effectInvolvesEquipmentEntity.some(
                          equipment onto.hasMaterialTransferEquipment.some(equipment onto.CentrifugalPump) &
                          equipment_onto.hasOperationMode.some(equipment_onto.StartUpOperation)) &
                      effectImpliedByCause.some(causes_onto.WrongMountingOfNonReturnValve)))]

class Overheating(Effect):
    equivalent_to = [Effect &
                     ((effectInvolvesEquipmentEntity.some(equipment onto.CompressorEntity) &
                       isEffectOfDeviation.some(deviation onto.HighTemperature))
                     |
                     (effectInvolvesEquipmentEntity.some(equipment onto.CompressorEntity) &
                      effectImpliedByCause.some(causes onto.MalfunctionLubricationSystem))
                     |
                     (effectInvolvesEquipmentEntity.some(equipment_onto.PumpEntity &
                                                         equipment_onto.hasIntendedFunction.some(
                                                             process onto.DeliverConstantVolumeFlow)) &
                      isEffectOfDeviation.some(deviation onto.HighTemperature) &
                      effectOfPropagatedCause.value(True))
                     |
                      FluidCirculatesInsidePump)]

class HeatBuildUp(effect_onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     effect onto.effectImpliedByCause.some(BlockedReboilerLines)
                     ]

class ColumnFlooded(effect onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     ((effect_onto.isEffectOfDeviation.some(deviation_onto.HighLevel) &
                       effect_onto.effectInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                       effect onto.effectInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity))
                     |
                     (effect onto.effectImpliedByCause.some(causes onto.ExcessiveInflow) &
                      effect onto.effectInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                      effect onto.effectInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity)
                      ))]
```

Appendix Listing M-4: Class definitions of effect concepts Part IV

```
class LiquidSlugging(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.hasMaterialTransferEquipment.some(
                                                                    equipment_onto.PistonCompressor | equipment_onto.ScrewCompressor)) &
                                effect_onto.effectInvolvesSubstance.some(
                           substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium |
                                                               substance_onto.Refrigerant)) &
                                (effect_onto.effectImpliedByCause.some(causes_onto.ContaminationByWater |
                                                                       causes onto.OtherSubstanceFromUpstream)
                                 |
                                 effect onto.isEffectOfDeviation.some(deviation onto.OtherThanComposition)))]

LiquidSlugging.comment = ["Screw compressors have a higher tolerance to liquid slugging"]

class ExcessiveDischargeTemperature(effect_onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     ((effect onto.isEffectOfDeviation.some(deviation onto.HighTemperature) &
                       effect onto.effectInvolvesEquipmentEntity.some(equipment onto.CompressorEntity))
                        |
                      (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.CompressorEntity) &
                       effect onto.effectImpliedByCause.some(causes onto.MalfunctionLubricationSystem)))]

class IncreasedOilDischarge(effect onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     effect_onto.effectInvolvesSubstance.some(
                         substance onto.hasSpecificTask.some(substance onto.ProcessMedium |
                                                             substance onto.Refrigerant)) &
                     effect onto.isEffectOfDeviation.some(deviation onto.HighFlow) &
                     effect onto.effectInvolvesEquipmentEntity.some(equipment onto.CompressorEntity &
                                                                    equipment_onto.hasSubunit.some(equipment_onto.LubricationSystem))]

class IncompleteEvaporation(effect_onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((
                                effect onto.effectInvolvesSubstance.some(substance onto.hasSpecificTask.some(substance onto.Refrigerant)) &
                                effect onto.isEffectOfDeviation.some(deviation onto.HighFlow |
                                                                     deviation onto.LowTemperature) &
                                effect onto.effectInvolvesEquipmentEntity.some(equipment onto.FinTubeEvaporatorEntity |
                                                                               equipment_onto.ShellTubeEvaporatorEntity))
                      |
                      (effect onto.effectInvolvesSubstance.some(
                          substance onto.hasSpecificTask.some(substance onto.HeatingMedium)) &
                       effect onto.isEffectOfDeviation.some(deviation onto.LowTemperature) &
                       effect onto.effectInvolvesEquipmentEntity.some(equipment onto.ShellTubeEvaporatorEntity)))]

class IncompleteCondensation(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     (
                                effect_onto.effectInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.Refrigerant)) &
                                effect_onto.isEffectOfDeviation.some(deviation_onto.HighFlow |
                                                                     deviation onto.HighTemperature) &
                                effect onto.effectInvolvesEquipmentEntity.some(equipment onto.AirCooledCondenserEntity))]

class LossOfHeatTransfer(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.ShellTubeHeatExchangerEntity) &
                       effect_onto.effectImpliedByCause.some(causes_onto.Fouling))
                      |
                      (effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.AirCooledCondenserEntity) &
                       effect_onto.isEffectOfDeviation.some(deviation_onto.HighTemperature) &
                       effect onto.effectImpliedByCause.some(causes onto.WrongRotatingSpeed |
                                                             causes onto.Fouling |
                                                             causes onto.HighAmbientTemperature |
                                                             causes onto.NonCondensables))
                      |
                      (effect_onto.effectImpliedByCause.some(NoSteamFlow)))]

class ReducedHeatingCapacity(effect_onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     (effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.StabilizerColumnEntity) &
                      effect onto.isEffectOfDeviation.some(deviation onto.LowTemperature))]

class IncreasedHeatingCapacity(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     (effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity) &
                      effect_onto.effectImpliedByCause.some(causes_onto.MoreSteamFlow))]

class IncreasedWear(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((effect onto.effectInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                      equipment_onto.hasIntendedFunction.some(
                                                                          process_onto.DeliverConstantVolumeFlow)) &
                       effect_onto.effectImpliedByCause.some(OperationBelowMinimumFlowRate)))]

class PumpRunningDry(effect_onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((effect onto.effectInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                      equipment onto.hasIntendedFunction.some(
                                                                          process onto.DeliverConstantVolumeFlow)) &
                       effect_onto.effectImpliedByCause.some(causes_onto.ClosedInletValve |
                                                             causes_onto.LossOfInflow |
                                                             causes_onto.BlockedInflowLine)))]

PumpRunningDry.comment = [
    "https://www.worldpumps.com/operating-design/features/how-to-overcome-the-challenge-of-dry-running/"]

class PumpDeliversNoLiquid(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process_onto.DeliverConstantVolumeFlow)) &
                      effect_onto.effectImpliedByCause.some(causes_onto.MissingImpeller |
                                                            causes_onto.EntrainedAir) &
                      effect_onto.isEffectOfDeviation.some(deviation_onto.NoFlow))
                     ]

class Cavitation(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((effect_onto.effectImpliedByCause.some(InsufficientNPSH) &
                       effect_onto.effectInvolvesEquipmentEntity.some(
                           equipment_onto.hasIntendedFunction.some(process_onto.DeliverConstantVolumeFlow)))
                      |
                      (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.PumpEntity &
                                                                      equipment onto.hasIntendedFunction.some(process onto.DeliverConstantVolumeFlow)) &
                       effect onto.effectOfPropagatedCause.some(True) &
                       effect onto.isEffectOfDeviation.some(deviation onto.HighTemperature)))]

class PoorPumpPerformance(effect onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     (effect_onto.effectImpliedByCause.some(EntrainedAir | causes_onto.ImpellerFault) &
                      effect onto.isEffectOfDeviation.some(deviation onto.LowFlow))
                     ]
```

Appendix Listing M-5: Class definitions of effect concepts Part V

```
class RunawayReaction(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((effect onto.effectInvolvesEquipmentEntity.some(equipment onto.ReactorEntity) &
                       (effect_onto.effectImpliedByCause.some(causes_onto.CoolingFailure |
                                                              causes_onto.ConfusionOfSubstances |
                                                              causes_onto.NoFeed |
                                                              causes_onto.Pollution |
                                                              ChargingFailure |
                                                              DosingFailure
                                                              ))
                      |
                      (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.ReactorEntity) &
                       effect onto.isEffectOfDeviation.some(deviation onto.ReverseFlow))
                     ))]

class InsufficientAmountOfLiquidRefrigerant(effect_onto.Effect):
    equivalent to = [effect onto.Effect &
                     (effect onto.isEffectOfDeviation.some(deviation onto.OtherThanComposition |
                                                           deviation onto.LowLevel) &
                      effect onto.effectInvolvesEquipmentEntity.some(equipment onto.PressureReceiverEntity &
                                                                     equipment onto.hasIntendedFunction.some(
                                                                         process onto.ModeIndependent)) &
                      effect onto.effectInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.Refrigerant))
                      )
                     ]

class ScrubberAgentNotAvailable(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     ((effect onto.effectInvolvesSubstance.some(
                         substance onto.hasSpecificTask.some(substance onto.ScrubbingAgent)) &
                       effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.WetScrubberEntity &
                                                                      equipment_onto.hasFixture.some(
                                                                  equipment_onto.LiquidDistributor) &
                                                                      equipment_onto.hasIntendedFunction.some(
                                                                          process onto.ModeIndependent)) &
                       effect onto.effectInvolvesSubstance.some(
                           substance onto.hasStateOfAggregation.some(substance onto.Liquid)) &
                       effect onto.isEffectOfDeviation.some(deviation onto.NoFlow))
                      |
                      ((effect_onto.effectInvolvesSubstance.some(
                          substance_onto.hasSpecificTask.some(substance_onto.ScrubbingAgent)) &
                        effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.WetScrubberEntity &
                                                                       equipment_onto.hasIntendedFunction.some(
                                                                           process_onto.ModeIndependent)) &
                        effect onto.effectImpliedByCause.some(LossOfInflow |
                                                              BlockedInflowLine) &
                        effect onto.effectInvolvesSubstance.some(
                            substance onto.hasStateOfAggregation.some(substance onto.Liquid)))))
                     ]

class InsufficientGasPurification(effect_onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     (effect_onto.effectInvolvesEquipmentEntity.some(equipment_onto.WetScrubberEntity &
                                                                     equipment_onto.hasIntendedFunction.some(
                                                                         process onto.ModeIndependent)) &
                      effect onto.effectInvolvesSubstance.some(
                          substance onto.hasStateOfAggregation.some(substance onto.Liquid |
                                                                    substance onto.Multiphase)) &
                      effect_onto.isEffectOfDeviation.some(deviation_onto.OtherThanComposition))
                     ]

class FloodedPackedBed(effect_onto.Effect):
    equivalent_to = [effect_onto.Effect &
                     (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.WetScrubberEntity &
                                                                     equipment onto.hasIntendedFunction.some(
                                                                         process onto.ModeIndependent)) &
                      effect onto.effectInvolvesSubstance.some(
                          substance_onto.hasStateOfAggregation.some(substance_onto.Liquid)) &
                      effect_onto.isEffectOfDeviation.some(deviation_onto.HighLevel))
                     ]

class AbnormalOperationCondition(effect onto.Effect):
    equivalent to = [effect onto.Effect &
                     (effect onto.effectInvolvesEquipmentEntity.some(equipment onto.WetScrubberEntity &
                                                                     equipment onto.hasIntendedFunction.some(
                                                                         process onto.ModeIndependent)) &
                      effect onto.isEffectOfDeviation.some(deviation onto.HighTemperature |
                                                           deviation_onto.LowTemperature |
                                                           deviation_onto.LowPressure |
                                                           deviation onto.HighPressure |
                                                           deviation onto.HighFlow |
                                                           deviation onto.LowFlow |
                                                           deviation onto.HighLevel |
                                                           deviation onto.OtherThanComposition))
                     ]
```

Appendix Listing M-6: Class definitions of effect concepts Part VI

Appendix N Formal ontology class definitions consequences

```
class PumpBreakdown(Consequence):
    equivalent_to = [Consequence &
                     ((isConsequenceOfEffect.some(effect_onto.IncreasedWear |
                                                  effect_onto.Cavitation |
                                                  effect_onto.PumpRunningDry))
                      |
                      # Because of this restriction DangerOfBleve and PumpBreakdown do not occur simulatenously
                      (isConsequenceOfEffect.some(effect_onto.Overheating) & isConsequenceOfDeviation.some(deviation_onto.HighTemperature)))]

PumpBreakdown.comment = ["'Failure' when equipment condition reaches an unacceptable level but still operating",
                         "'Breakdown' not functioning anymore",
                         "There is also specific definition of the concept in the compressor_onto"]

class CompressorBreakdown(Consequence):
    equivalent_to = [Consequence &
                     (consequenceInvolvesEquipmentEntity.some(equipment_onto.CompressorEntity) &
                      isConsequenceOfEffect.some(effect_onto.Overheating | effect_onto.ExcessiveDischargeTemperature |
                                                 effect_onto.IncreasedOilDischarge | effect_onto.LiquidSlugging))]

class ProductionDowntime(Consequence):
    equivalent_to = [Consequence &
                     (isConsequenceOfEffect.some(effect_onto.CompressorNotOperating | effect_onto.PumpDeliversNoLiquid |
                                                 effect_onto.InsufficientFilling | effect_onto.EmptyingOfContainer)
                            |
                            (isConsequenceOfDeviation.some(deviation_onto.NoFlow) & consequenceInvolvesEquipmentEntity.some(equipment_onto.Sink-
Entity))
                            |
                            ((isConsequenceOfDeviation.some(deviation_onto.LowLevel) &
                              consequenceInvolvesEquipmentEntity.some(
                                  equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank | equipment_onto.PressureVessel))
                              ) &
                             isSubsequentConsequence.some(PumpBreakdown))
                            |
                            isSubsequentConsequence.some(PumpBreakdown | CompressorBreakdown)
                            |
                            (consequenceImpliedByCause.some(causes_onto.BlockedOutflowLine) &
                             isConsequenceOfDeviation.some(deviation_onto.NoFlow) &
                             consequenceInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity))
                            |
                            (consequenceImpliedByCause.some(causes_onto.LossOfInflow | causes_onto.ValveWronglyClosed |
                                                            causes_onto.IncorrectFilling | causes_onto.ClosedInletValve) &
                             isConsequenceOfEffect.some(effect_onto.EmptyingOfContainer))
                            |
                            (isConsequenceOfEffect.some(effect_onto.LossOfHeatTransfer) &
                             consequenceInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity)))]

ProductionDowntime.comment = ["There is also specific definition of the concept in compressor_onto"]

class ReductionOfCoolingCapacity(Consequence):
    equivalent_to = [Consequence &
                     (isConsequenceOfEffect.some(effect_onto.LossOfHeatTransfer) &
                      consequenceInvolvesSubstance.some(substance_onto.hasSpecificTask.some(substance_onto.Refrigerant)))]

class PROPAGATED CONSEQUENCE(Consequence):
    equivalent_to = [Consequence &
                     ((consequenceImpliedByCause.some(causes_onto.IncorrectSetPointControlValve | causes_onto.ConfusionOfSubstances |
                         causes_onto.ValveWronglyOpened | causes_onto.InadvertentContamination | causes_onto.BypassOpened |
                         causes_onto.ReducedFlowArea))
                      |
                      (consequenceImpliedByCause.some(causes_onto.ExcessiveInflow) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.InertgasBlanketingEntity))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.HighPressure | deviation_onto.LowFlow | deviation_onto.HighFlow |
                                                     deviation_onto.NoFlow | deviation_onto.OtherThanComposition) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.InertgasBlanketingEntity))
                      |
                      (consequenceImpliedByCause.some(causes_onto.WrongTankLinedUp) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.TankTruckEntity))
                      |
                      (isConsequenceOfEffect.some(effect_onto.IncompleteEvaporation | effect_onto.AbnormalEvaporation) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.LowTemperature) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.HighPressure) & consequenceImpliedByCause.some(causes_onto.InternalLeakage))
                      |
                      (consequenceImpliedByCause.some(causes_onto.IncreasedInletPressure) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.InertgasBlanketingEntity))
                      |
                      (isConsequenceOfEffect.some(effect_onto.InsufficientInertization) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.InertgasBlanketingEntity))
                      |
                      (consequenceInvolvesEquipmentEntity.some(equipment_onto.hasPiping.some(equipment_onto.TankTruckHose)) &
                       consequenceImpliedByCause.some(causes_onto.ContaminationInUnloadingLines))
                      |
                      (consequenceInvolvesEquipmentEntity.some(equipment_onto.PumpEntity) &
                       consequenceImpliedByCause.some(causes_onto.PumpIncorrectlySet | causes_onto.WrongImpeller))
                      |
                      (consequenceInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity) &
                       isConsequenceOfEffect.some(effect_onto.IncreasedHeatingCapacity))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.NoFlow) &
                       consequenceImpliedByCause.some(causes_onto.ValveWronglyClosed | causes_onto.ValveIntactUnintentionallyClosed |
                                                      causes_onto.ClosedInletValve | causes_onto.MissingImpeller |
                                                      causes_onto.ImpellerFault | causes_onto.EntrainedAir))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.OtherThanComposition) &
                       consequenceImpliedByCause.some(causes_onto.EntrainedAir))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.NoFlow | deviation_onto.LowFlow) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.SourceEntity | equipment_onto.ConnectionPipeEntity))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.HighPressure | deviation_onto.HighTemperature) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.SteamDrivenReboilerEntity))
                      |
                      (isConsequenceOfDeviation.some(deviation_onto.NoFlow) &
                       consequenceInvolvesEquipmentEntity.some(equipment_onto.ValveEntity) &
                       consequenceImpliedByCause.some(causes_onto.WrongMountingOfNonReturnValve)))
                     |
                     (consequenceInvolvesEquipmentEntity.some(equipment_onto.SourceEntity) &
                      isConsequenceOfDeviation.some(deviation_onto.OtherThanComposition) &
                      consequenceImpliedByCause.some(causes_onto.InadvertentContamination |
                                                     causes_onto.OtherSubstanceFromUpstream))
                     ]
```

Appendix Listing N-1: Class definitions of consequence concepts Part I

```
class PoorProductQuality(Consequence):
    equivalent_to = [Consequence &
                     ((consequenceInvolvesSubstance.some(
                         substance_onto.hasSpecificTask.some(substance_onto.ProcessMedium)) &
                       consequenceInvolvesEquipmentEntity.some((equipment_onto.StorageTankEntity |
                                                                equipment_onto.PressureReceiverEntity |
                                                                equipment_onto.StabilizerColumnEntity |
                                                                equipment_onto.DistillationColumnEntity |
                                                                equipment onto.PlateHeatExchangerEntity |
                                                                equipment onto.PressureVesselEntity |
                                                                equipment onto.ShellTubeHeatExchangerEntity |
                                                                equipment onto.ShellTubeEvaporatorEntity) &
                                                                equipment onto.hasIntendedFunction.some(
                                                                    process_onto.ModeIndependent)) &
                       consequenceImpliedByCause.some(causes_onto.InadvertentContamination))
                     |
                     (isConsequenceOfEffect.some(effect onto.BacteriaGrowth) &
                      consequenceInvolvesEquipmentEntity.some(
                          equipment onto.hasIntendedFunction.some(process onto.Storing)))
                     |
                     (consequenceImpliedByCause.some(causes onto.InadvertentContamination) &
                      consequenceInvolvesEquipmentEntity.some(
                          equipment onto.hasIntendedFunction.some(process onto.Filling)))
                     |
                     (isConsequenceOfEffect.some(effect_onto.AccumulationOfImpurities) &
                      consequenceInvolvesEquipmentEntity.some(
                          equipment onto.hasIntendedFunction.some(process onto.Filling |
                                                                  process onto.ModeIndependent)))
                     |
                     (consequenceInvolvesSubstance.some(
                         substance onto.hasSpecificTask.some(substance onto.ProcessMedium)) &
                      consequenceInvolvesEquipmentEntity.some(equipment_onto.SettlingTankEntity &
                                                              equipment_onto.hasIntendedFunction.some(
                                                                  process_onto.Separating)) &
                      isConsequenceOfEffect.some(effect_onto.PoorSeparation))
                     |
                     (isConsequenceOfDeviation.some(deviation onto.OtherThanComposition) &
                      consequenceImpliedByCause.some(causes onto.InternalLeakage))
                     )]

class EmergenceOfIgnitionSource(Consequence):
    equivalent_to = [Consequence &
                     isConsequenceOfEffect.some(effect_onto.GenerationOfElectrostaticCharge)]

class LossOfPrimaryContainment(Consequence):
    equivalent_to = [Consequence &
                     ((isConsequenceOfEffect.some(effect onto.FatigueFracture |
                                                  effect onto.Fracture |
                                                  effect onto.LossOfMechanicalIntegrity |
                                                  effect_onto.PotentialViolentReactionWithOxidizers |
                                                  effect_onto.BrittleFracture |
                                                  effect_onto.DrainlineFracture |
                                                  effect_onto.GasDispersion |
                                                  effect_onto.PoolFormation)
                      )
                      |
                      (isConsequenceOfEffect.some(effect onto.Overfilling) &
                       consequenceInvolvesEquipmentEntity.some(
                           equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank))))
                     ]

LossOfPrimaryContainment.comment = ["LOPC is defined in API Guide to Reporting Process Safety Events, Version 3.0"]

class FireHazard(Consequence):
    equivalent_to = [Consequence &
                     (consequenceRequiresBoundaryCondition.some(boundary onto.SufficientOxygenAvailable) &
                      consequenceRequiresBoundaryCondition.some(boundary onto.DirectIgnition) &
                      isSubsequentConsequence.some(LossOfPrimaryContainment) &
                      consequenceInvolvesSubstance.some(
                          substance_onto.hasHazardClass.some(
                              substance_onto.FlammableLiquidCategory1 |
                              substance_onto.FlammableLiquidCategory2 |
                              substance onto.FlammableLiquidCategory3
                          )))]

class RiskOfExplosiveAtmosphere(Consequence):
    equivalent_to = [Consequence &
                     (
                             (consequenceRequiresBoundaryCondition.some(boundary_onto.LocatedOutside |
                                                                        boundary_onto.SufficientOxygenAvailable) &
                              consequenceRequiresBoundaryCondition.some(boundary onto.DelayedIgnition) &
                              isSubsequentConsequence.some(LossOfPrimaryContainment) &
                              consequenceInvolvesSubstance.some(
                                  substance onto.hasHazardClass.some(
                                      substance onto.FlammableLiquidCategory1 |
                                      substance onto.FlammableLiquidCategory2 |
                                      substance onto.FlammableLiquidCategory3 |
                                      substance_onto.FlammableGasCategory1 |
                                      substance_onto.FlammableGasCategory2 |
                                      substance_onto.PyrophoricGasCategory1 |
                                      substance onto.AerosolCategory1 |
                                      substance onto.AerosolCategory2
                                  )
                              ))
                             |
                             (consequenceRequiresBoundaryCondition.some(boundary_onto.LocatedOutside |
                                                                        boundary_onto.SufficientOxygenAvailable) &
                              isSubsequentConsequence.some(LossOfPrimaryContainment) &
                              consequenceRequiresBoundaryCondition.some(boundary_onto.DelayedIgnition) &
                              consequenceInvolvesSubstance.some(
                                  substance onto.hasStabilityReactivityInformation.some(
                                      substance onto.FormsExplosiveMixtureWithAir
                                  )))
                             |
                             (isConsequenceOfEffect.some(effect_onto.InsufficientInertization) &
                              consequenceInvolvesEquipmentEntity.some(equipment_onto.AtmosphericStorageTank | equipment_onto.SettlingTankEntity) &
                              consequenceRequiresBoundaryCondition.some(boundary_onto.IntroductionOfAir) &
                              consequenceInvolvesSubstance.some(
                                  substance_onto.hasHazardClass.some(
                                      substance_onto.FlammableLiquidCategory1 |
                                      substance onto.FlammableLiquidCategory2 |
                                      substance onto.FlammableLiquidCategory3 |
                                      substance onto.FlammableGasCategory1 |
                                      substance onto.FlammableGasCategory2 |
                                      substance_onto.PyrophoricGasCategory1 |
                                      substance_onto.AerosolCategory1 |
                                      substance_onto.AerosolCategory2)))
                             |
                             (consequenceImpliedByCause.some(causes_onto.InternalLeakage) &
                              consequenceInvolvesSubstance.some(
                                  substance onto.hasStabilityReactivityInformation.some(substance onto.FormsExplosiveMixturesWithOxidizingAgents))))]

RiskOfExplosiveAtmosphere.comment = ["Eindringen luft, und betriebsdruck kleiner umgebungsdruck",
                                     "https://www.ketopumps.com/media/1342/keto-green-paper-centrifugal-pump-explosions.pdf"]
```

Appendix Listing N-2: Class definitions of consequence concepts Part II

```
class DangerOfBleve(Consequence):
    equivalent to = [Consequence &
                     (isConsequenceOfEffect.some(effect onto.FluidCirculatesInsidePump) &
                      consequenceImpliedByCause.some(causes_onto.DeadHeadingOfPump) &
                      consequenceInvolvesEquipmentEntity.some(equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump)))]

DangerOfBleve.comment = ["physical explosion"]

class NoStabilization(consequence onto.Consequence):
    equivalent to = [consequence onto.Consequence &
                     consequence onto.consequenceInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity) &
                     consequence onto.isConsequenceOfEffect.some(effect onto.LossOfHeatTransfer)]


class PoorStripping(consequence_onto.Consequence):
    equivalent_to = [consequence_onto.Consequence & consequence_onto.isConsequenceOfEffect.some(ColumnFlooded)]


class PoorStabilization(consequence onto.Consequence):
    equivalent to = [consequence onto.Consequence &
                     (consequence onto.consequenceInvolvesEquipmentEntity.some(equipment onto.StabilizerColumnEntity) &
                      consequence onto.isConsequenceOfEffect.some(effect onto.ReducedHeatingCapacity |
                                                                  effect onto.LossOfHeatTransfer |
                                                                  effect_onto.IncompleteEvaporation)
                                 )]

class ReductionOfCoolingCapacity(consequence onto.Consequence):
    equivalent to = [consequence onto.Consequence &
                     ((consequence onto.isConsequenceOfEffect.some(LossOfHeatTransfer |
                                                                   IncompleteEvaporation) &
                       consequence onto.consequenceInvolvesSubstance.some(substance onto.hasSpecificTask.some(substance onto.Refrigerant)))
                     |
                      (
                                 consequence_onto.consequenceInvolvesEquipmentEntity.some(equipment_onto.PressureReceiverEntity) &
                                 consequence_onto.consequenceInvolvesSubstance.some(
                                                                     substance onto.hasSpecificTask.some(substance onto.Refrigerant)) &
                                 consequence onto.isConsequenceOfEffect.some(effect onto.AbnormalEvaporation))
                      )]
```

Appendix Listing N-3: Class definitions of consequence concepts Part III

Appendix O Formal ontology class definitions risk assessment

```
class Likelihood(Thing):
    pass

class likelihoodInvolvesCause(Likelihood >> causes onto.Cause):
    pass

class likelihoodInvolvesUnderlyingcause(Likelihood >> causes onto.UnderlyingCause):
    pass

class likelihoodInvolvesEquipment(Likelihood >> equipment onto.EquipmentEntity):
    pass

class likelihoodInvolvesDeviation(Likelihood >> deviation_onto.Deviation):
    pass

class likelihoodRequiresBoundaryCondition(Likelihood >> boundary_onto.BoundaryCondition):
    pass

class likelihoodInvolvesSiteInformation(Likelihood >> site information.AmbientInformation):
    pass

class SeverityCategory(Thing):
    pass

class isSeverityOfConsequence(SeverityCategory >> consequence_onto.Consequence):
    pass

class severityInvolvesSubstance(SeverityCategory >> substance onto.Substance):
    pass

class severityInvolvesEquipment(SeverityCategory >> equipment_onto.EquipmentEntity):
    pass

class severityRequiresBoundaryCondition(SeverityCategory >> boundary_onto.BoundaryCondition):
    pass

class RiskCategory(Thing):
    pass

class involvesSeverity(RiskCategory >> SeverityCategory):
    pass

class involvesLikelihood(RiskCategory >> Likelihood):
    pass

class VeryUnlikely(Likelihood):
    pass
VeryUnlikely.comment = ["corresponds to category F5", "10^-6 - 10^-4"]

class Unlikely(Likelihood):
    equivalent_to = [Likelihood &
                     (
                             (likelihoodInvolvesUnderlyingcause.some(causes onto.ValveFailure) &
                              likelihoodInvolvesCause.some(causes onto.ClosedInletValve))
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes onto.ValveFailure) &
                              likelihoodInvolvesCause.some(causes onto.ClosedOutletValve))
                             |
                             likelihoodInvolvesCause.some(causes onto.BlockedOutflowLine)
                             |
                             (likelihoodInvolvesCause.some(causes_onto.MechanicalFailureOfSupport) &
                              likelihoodInvolvesEquipment.some(
                                  equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank)))
                             |
                             likelihoodInvolvesUnderlyingcause.some(causes onto.AbnormallyHotIntake)
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes onto.SolarRadiation) &
                              likelihoodInvolvesCause.some(causes onto.ThermalExpansion |
                                                           causes_onto.AbnormalHeatInput) &
                              likelihoodInvolvesDeviation.some(deviation_onto.HighPressure) &
                              likelihoodInvolvesEquipment.some(
                                  equipment onto.hasApparatus.some(equipment onto.PressureVessel))))]

Unlikely.comment = ["corresponds to category F4", "10^-4 - 10^-3"]

class Possible(Likelihood):
    equivalent_to = [Likelihood &
                     (((likelihoodInvolvesEquipment.some(
                                 equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.ScrewCompressor | equipment_onto.PistonCompressor)) &
                              likelihoodInvolvesCause.some(causes_onto.MalfunctionLubricationSystem))
                             |
                             (likelihoodInvolvesEquipment.some(
                                 equipment onto.hasMaterialTransferEquipment.some(equipment onto.PistonCompressor | equipment onto.ScrewCompressor |
                                                                                  equipment onto.CentrifugalPump | equipment onto.ReciprocatingPump)) &
                              likelihoodInvolvesDeviation.some(deviation onto.HighTemperature))
                             |
                             (likelihoodInvolvesEquipment.some(
                                 equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump | equipment_onto.ReciprocatingPump)) &
                              likelihoodInvolvesCause.some(causes_onto.DeadHeadingOfPump | causes_onto.OperationBelowMinimumFlowRate))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.ExcessiveInflow | causes_onto.PumpIncorrectlySet) &
                              likelihoodInvolvesEquipment.some(
                                  equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank | equipment onto.PressureVessel |
                                                                   equipment onto.OpenVessel)))
                             |
                             likelihoodInvolvesCause.some(causes_onto.PhysicalImpact | causes_onto.ControlValveFailsOpen | causes_onto.LossOfInflow |
                                                          causes_onto.WrongTankLinedUp | causes_onto.LeakingDrainValve |
                                                          causes_onto.ContaminationByWaterAndTemperatureFallsBelowFreezingPoint |
                                                          causes_onto.PumpingAgainstPolymerizedLine)
                             |
                             (likelihoodInvolvesCause.some(causes onto.MechanicalFailureOfSupport) &
                              likelihoodInvolvesEquipment.some(equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank)) &
                              likelihoodInvolvesSiteInformation.some(site information.DangerOfSeismicActivity))
                             |
                             (likelihoodInvolvesDeviation.some(deviation_onto.HighCorrosion) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.CondensationAirHumidity) &
                              likelihoodInvolvesEquipment.some(equipment_onto.StorageTankEntity))
                             |
                             (likelihoodInvolvesCause.some(causes onto.BlockedInflowLine) &
                              likelihoodInvolvesUnderlyingcause.some(causes onto.DepositionOfImpurities))
                             |
                             (likelihoodInvolvesCause.some(causes onto.ThermalExpansion) &
                              likelihoodInvolvesUnderlyingcause.some(causes onto.BlockedPipingAndHeatInput))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.AbnormalHeatInput) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.SolarRadiation))
                             |
                             (likelihoodInvolvesDeviation.some(deviation onto.OtherThanComposition) &
                              likelihoodInvolvesCause.some(causes onto.OtherSubstanceFromUpstream)))]
Possible.comment = ["corresponds to category F3", "10^-3 - 10^-2"]
```

Appendix Listing O-1: Formal risk assessment classes; formal likelihood class definitions – Part I

```
class Occasional(Likelihood):
    equivalent_to = [Likelihood &
                     (likelihoodInvolvesCause.some(causes_onto.PumpSealFailure)
                             |
                             (likelihoodInvolvesEquipment.some(
                                 equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank |
                                                                  equipment_onto.OpenVessel |
                                                                  equipment_onto.PressureVessel)) &
                              likelihoodInvolvesCause.some(causes_onto.ExcessiveInflow |
                                                           causes_onto.IncorrectIndicationOfFillingLevel))
                             |
                             (likelihoodInvolvesEquipment.some(
                                 equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump)) &
                              likelihoodInvolvesDeviation.some(deviation_onto.HighVibration))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.WaterHammer) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.RapidlyClosingValve))
                             |
                             (likelihoodInvolvesEquipment.some(
                                 equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump)) &
                              likelihoodInvolvesCause.some(causes_onto.EntrainedAir |
                                                           causes_onto.ImpellerFault))
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.ExternalFire))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.MissingImpeller) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.MaintenanceError))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.PumpIncorrectlySet) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError))
                             |
                             (likelihoodInvolvesEquipment.some(
                                 equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump)) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.SuddenlyStoppingPump | causes_onto.SuddenStartingPump))
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.CondensationAirHumidity) &
                              likelihoodInvolvesCause.some(causes_onto.ContaminationByWater) &
                              likelihoodInvolvesEquipment.some(equipment_onto.StorageTankEntity))
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.LongStorageTimeOfStabilizer) &
                              likelihoodInvolvesCause.some(causes_onto.TooLittleStabilizer))
                             |
                              likelihoodInvolvesCause.some(causes_onto.ValveIntactUnintentionallyClosed |
                                                           causes_onto.InsufficientNPSH |
                                                           causes_onto.ReducedDwellTime)
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError) &
                              likelihoodInvolvesCause.some(causes_onto.DrainValveInadvertentlyOpened))
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError) &
                              likelihoodInvolvesEquipment.some(equipment_onto.PumpEntity) &
                              likelihoodInvolvesCause.some(causes_onto.ClosedInletValve))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.ValveWronglyOpened) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError))
                             |
                             likelihoodInvolvesCause.some(causes_onto.DeadHeadingOfPump)
                             |
                             (likelihoodInvolvesCause.some(causes_onto.InadvertentContamination) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.IntroductionOfRainwater | causes_onto.ContaminationInTankTruck))
                             |
                             likelihoodInvolvesCause.some(causes_onto.InadvertentContamination)
                             |
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.IntroductionOfRainwater) &
                              likelihoodInvolvesCause.some(causes_onto.ContaminationInUnloadingLines)))]

Occasional.comment = ["corresponds to category F2", "10^-2 - 10^-1"]

class Likely(Likelihood):
    equivalent_to = [Likelihood &
                     ((likelihoodInvolvesCause.some(causes_onto.ExternalLeakage) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.HoseIncorrectlyConnected |
                                                                     causes_onto.BrokenHose |
                                                                     causes_onto.MaintenanceError |
                                                                     causes_onto.LossOfLeakTightness))
                             |
                             likelihoodInvolvesUnderlyingcause.some(causes_onto.FailureControlLoop |
                                                                    causes_onto.MalfunctionPressureController |
                                                                    causes_onto.PressureIndicatorControllerFailure |
                                                                    causes_onto.FlowIndicatorControllerFailure |
                                                                    causes_onto.MalfunctionFlowController |
                                                                    causes_onto.MalfunctionControlAir |
                                                                    causes_onto.LevelIndicatorControllerFailure |
                                                                    causes_onto.PowerFailure)
                             |
                             likelihoodInvolvesCause.some(causes_onto.NoInertgasSupply |
                                                          causes_onto.ExcessiveFluidWithdrawal |
                                                          causes_onto.PumpOperationFailure)
                             |
                             (likelihoodInvolvesCause.some(causes_onto.IncorrectSetPointControlValve) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.ValveClosedPressureBuildUpInPiping) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.ExcessiveInflow) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.BypassOpened) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.OperationalError))
                             |
                             (likelihoodRequiresBoundaryCondition.some(boundary_onto.LocatedOutside) &
                              likelihoodInvolvesUnderlyingcause.some(causes_onto.SolarRadiation) &
                              likelihoodInvolvesEquipment.some(equipment_onto.StorageTankEntity) &
                              likelihoodInvolvesCause.some(causes_onto.ThermalExpansion)))]

Likely.comment = ["corresponds to category F1", "10^-1 - 1^0"]

class VeryLikely(Likelihood):
    equivalent_to = [Likelihood &
                     (
                             (likelihoodInvolvesUnderlyingcause.some(causes_onto.AmbientTemperatureChange))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.InsufficientThermalOutbreathing) &
                              likelihoodInvolvesEquipment.some(equipment_onto.hasApparatus.some(
                                  equipment_onto.AtmosphericStorageTank)))
                             |
                             (likelihoodInvolvesCause.some(causes_onto.LiquidTransferWithoutCompensation) &
                              likelihoodInvolvesEquipment.some(equipment_onto.hasApparatus.some(
                                  equipment_onto.AtmosphericStorageTank))))]

VeryLikely.comment = ["corresponds to category F0", "> 1 p.a."]
```

Appendix Listing O-2: Formal likelihood class definitions – Part II

```
class Catastrophic(SeverityCategory):
    equivalent_to = [SeverityCategory &
                     ((isSeverityOfConsequence.some(consequence onto.LossOfPrimaryContainment) &
                              severityRequiresBoundaryCondition.some(
                                  boundary_onto.SeveralPeoplePresentInTheNearField))
                             |
                             (isSeverityOfConsequence.some(consequence_onto.LossOfPrimaryContainment) &
                              severityRequiresBoundaryCondition.some(
                                  boundary onto.SeveralPeoplePresentInTheNearField))
                             |
                             (isSeverityOfConsequence.some(consequence onto.LossOfPrimaryContainment) &
                              severityRequiresBoundaryCondition.some(
                                  boundary onto.SeveralPeoplePresentInTheNearField) &
                              severityInvolvesSubstance.some(substance_onto.hasHazardClass.some(
                                  substance_onto.PyrophoricGasCategory1))
                              ))]

Catastrophic.comment = ["corresponds to category S0"]

class Severe(SeverityCategory):
    equivalent_to = [SeverityCategory &
                     ((isSeverityOfConsequence.some(consequence onto.DangerOfBleve) &
                              severityInvolvesEquipment.some(equipment onto.hasMaterialTransferEquipment.some(
                                  equipment_onto.CentrifugalPump)) &
                              severityRequiresBoundaryCondition.some(boundary_onto.PersonnelPresentInTheNearField))
                             |
                             (severityRequiresBoundaryCondition.some(boundary onto.PersonnelPresentInTheNearField) &
                              severityInvolvesSubstance.some(substance onto.hasHazardClass.some(
                                  substance onto.FlammableLiquidCategory1 |
                                  substance onto.FlammableLiquidCategory2)) &
                              isSeverityOfConsequence.some(consequence onto.LossOfPrimaryContainment))
                             |
                             (severityInvolvesEquipment.some(equipment_onto.StorageTankEntity) &
                              severityRequiresBoundaryCondition.some(boundary_onto.PersonnelPresentInTheNearField) &
                              isSeverityOfConsequence.some(consequence_onto.LossOfPrimaryContainment) &
                              severityInvolvesSubstance.some(substance onto.hasHazardClass.some(
                                  substance onto.FlammableLiquidCategory1 |
                                  substance onto.FlammableLiquidCategory2)))
                             |
                             (severityRequiresBoundaryCondition.some(boundary_onto.PersonnelPresentInTheNearField) &
                              isSeverityOfConsequence.some(consequence_onto.EmergenceOfIgnitionSource) &
                              severityInvolvesSubstance.some(substance_onto.hasHazardClass.some(
                                  substance_onto.FlammableLiquidCategory1 |
                                  substance_onto.FlammableLiquidCategory2
                              )))
                             |
                             (isSeverityOfConsequence.some(consequence onto.LossOfPrimaryContainment) &
                              severityRequiresBoundaryCondition.some(boundary onto.PersonnelPresentInTheNearField) &
                              severityInvolvesSubstance.some(substance onto.hasHazardClass.some(
                                  substance_onto.SpecificTargetOrganToxicitySingleExposureCategory1 |
                                  substance_onto.SpecificTargetOrganToxicityRepeatedExposureCategory1 |
                                  substance_onto.AspirationHazardCategory1 |
                                  substance_onto.ReproductiveToxicityCategory1 |
                                  substance_onto.SkinCorrosionIrritationCategory1
                              )))
                             |
                             (severityRequiresBoundaryCondition.some(boundary onto.PersonnelPresentInTheNearField) &
                              isSeverityOfConsequence.some(consequence onto.RiskOfExplosiveAtmosphere) &
                              severityInvolvesSubstance.some(substance onto.hasHazardClass.some(
                                  substance_onto.FlammableLiquidCategory1 |
                                  substance_onto.FlammableLiquidCategory2 |
                                  substance_onto.FlammableGasCategory1 |
                                  substance_onto.FlammableGasCategory2
                              )))]

Severe.comment = ["corresponds to category S1"]

class Serious(SeverityCategory):
    equivalent_to = [SeverityCategory &
                     (isSeverityOfConsequence.some(consequence_onto.LossOfPrimaryContainment) &
                      severityInvolvesSubstance.some(substance_onto.hasHazardClass.some(
                          substance_onto.SpecificTargetOrganToxicitySingleExposureCategory3 |
                          substance onto.SpecificTargetOrganToxicityRepeatedExposureCategory2 |
                          substance onto.SpecificTargetOrganToxicitySingleExposureCategory2 |
                          substance onto.SpecificTargetOrganToxicityRepeatedExposureCategory3 |
                          substance onto.ReproductiveToxicityCategory2 |
                          substance onto.FlammableLiquidCategory3 |
                          substance onto.SkinCorrosionIrritationCategory2 |
                          substance_onto.SkinCorrosionIrritationCategory3
                      )))]

Serious.comment = ["corresponds to category S2"]

class Significant(SeverityCategory):
    pass

Significant.comment = ["corresponds to category S3"]

class Minor(SeverityCategory):
    equivalent_to = [SeverityCategory &
                     (
                         (isSeverityOfConsequence.some(consequence onto.ProductionDowntime) |
                          isSeverityOfConsequence.some(consequence onto.PoorProductQuality) |
                          isSeverityOfConsequence.some(consequence onto.PumpBreakdown) |
                          isSeverityOfConsequence.some(consequence onto.CompressorBreakdown))
                     )]

Minor.comment = ["corresponds to category S4"]
```

Appendix Listing O-3: Formal severity class definitions

```
# === Risk category definitions
class A(RiskCategory):
    equivalent to = [RiskCategory &
                     ((involvesLikelihood.some(VeryLikely) &
                       involvesSeverity.some(Catastrophic))
                     |
                     (involvesLikelihood.some(Likely) &
                      involvesSeverity.some(Catastrophic))
                     |
                     (involvesLikelihood.some(VeryLikely) &
                      involvesSeverity.some(Severe))
                     )]


class B(RiskCategory):
    equivalent_to = [RiskCategory &
                     ((involvesLikelihood.some(VeryLikely) &
                       involvesSeverity.some(Serious))
                     |
                     (involvesLikelihood.some(VeryLikely) &
                      involvesSeverity.some(Significant))
                     |
                     (involvesLikelihood.some(Likely) &
                      involvesSeverity.some(Serious))
                     |
                     (involvesLikelihood.some(Likely) &
                      involvesSeverity.some(Severe))
                     |
                     (involvesLikelihood.some(Occasional) &
                      involvesSeverity.some(Serious))
                     |
                     (involvesLikelihood.some(Occasional) &
                      involvesSeverity.some(Severe))
                     |
                     (involvesLikelihood.some(Occasional) &
                      involvesSeverity.some(Catastrophic))
                     |
                     (involvesLikelihood.some(Possible) &
                      involvesSeverity.some(Catastrophic))
                     |
                     (involvesLikelihood.some(Possible) &
                      involvesSeverity.some(Severe))
                     |
                     (involvesLikelihood.some(Unlikely) &
                      involvesSeverity.some(Catastrophic))
                     )]


class C(RiskCategory):
    equivalent_to = [RiskCategory &
                     ((involvesLikelihood.some(VeryLikely) &
                       involvesSeverity.some(Minor))
                     |
                     (involvesLikelihood.some(Likely) &
                      involvesSeverity.some(Minor))
                     |
                     (involvesLikelihood.some(Likely) &
                      involvesSeverity.some(Significant))
                     |
                     (involvesLikelihood.some(Occasional) &
                      involvesSeverity.some(Significant))
                     |
                     (involvesLikelihood.some(Possible) &
                      involvesSeverity.some(Serious))
                     |
                     (involvesLikelihood.some(Unlikely) &
                      involvesSeverity.some(Serious))
                     |
                     (involvesLikelihood.some(Unlikely) &
                      involvesSeverity.some(Severe))
                     |
                     (involvesLikelihood.some(VeryUnlikely) &
                      involvesSeverity.some(Catastrophic))
                     |
                     (involvesLikelihood.some(VeryUnlikely) &
                      involvesSeverity.some(Severe))
                     )]


class D(RiskCategory):
    equivalent to = [RiskCategory &
                     ((involvesLikelihood.some(Occasional) &
                       involvesSeverity.some(Minor))
                     |
                     (involvesLikelihood.some(Possible) &
                      involvesSeverity.some(Minor))
                     |
                     (involvesLikelihood.some(Possible) &
                      involvesSeverity.some(Significant))
                     |
                     (involvesLikelihood.some(Unlikely) &
                      involvesSeverity.some(Minor))
                     |
                     (involvesLikelihood.some(Unlikely) &
                      involvesSeverity.some(Significant))
                     |
                     (involvesLikelihood.some(VeryUnlikely) &
                      involvesSeverity.some(Serious))
                     |
                     (involvesLikelihood.some(VeryUnlikely) &
                      involvesSeverity.some(Significant))
                     |
                     (involvesLikelihood.some(VeryUnlikely) &
                      involvesSeverity.some(Minor))
                     )]
```

Appendix Listing O-4: Formal risk class definitions

Appendix P Formal ontology class definitions safeguards

```
class AddCorrosionInhibitor(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardOfDeviation.some(deviation_onto.HighCorrosion) &
                      safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity))]

class OverFlowValveAndKickBackLine(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsEffect.some(effect_onto.FluidCirculatesInsidePump | effect_onto.Overheating)]

class ImplementQuickConnectSystem(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsCause.some(causes_onto.WrongTankLinedUp) &
                     safeguardPreventsUnderlyingCause.some(causes_onto.OperationalError)]

class ImplementFrequentDrainingOff(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(equipment_onto.SettlingTankEntity) &
                     safeguardPreventsEffect.some(effect_onto.PoorSeparation)]

class OverfillProtection(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardDependsOnRiskCategory.some(risk_assessment_onto.B | risk_assessment_onto.A) &
                     safeguardPreventsEffect.some(effect_onto.Overfilling)]

class PressureReliefValve(Safeguard):
    equivalent_to = [Safeguard &
                     ((safeguardInvolvesEquipmentEntity.some(equipment_onto.PressureVesselEntity | equipment_onto.PumpEntity |
                                                             equipment_onto.ShellTubeHeatExchangerEntity) &
                       safeguardPreventsEffect.some(effect_onto.Fracture))
                      |
                      (safeguardInvolvesEquipmentEntity.some(equipment_onto.PressureVesselEntity |
                                                             equipment_onto.CompressorEntity |
                                                             equipment_onto.SettlingTankEntity |
                                                             equipment_onto.ShellTubeHeatExchangerEntity) &
                       safeguardOfDeviation.some(deviation_onto.HighPressure))
                      |
                      (safeguardPreventsEffect.some(effect_onto.UnintendedExothermicPolymerization) &
                       safeguardInvolvesEquipmentEntity.some(equipment_onto.PressureVesselEntity |
                                                             equipment_onto.SettlingTankEntity |
                                                             equipment_onto.ShellTubeHeatExchangerEntity |
                                                             equipment_onto.StorageTankEntity)))]

PressureReliefValve.comment = ["Must be certified for protecting pressure above 0.5 bar(g)"]

class AutomaticWaterDetectionSystem(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity) &
                     safeguardDependsOnRiskCategory.some(risk_assessment_onto.A | risk_assessment_onto.B) &
                     safeguardPreventsCause.some(causes_onto.ContaminationByWater)]

class ImplementVibrationDampener(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity | equipment_onto.CompressorEntity) &
                      safeguardOfDeviation.some(deviation_onto.HighVibration))]

class ImplementNoFlowAlarm(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect_onto.InsufficientInertization) &
                      safeguardPreventsCause.some(causes_onto.NoInertgasSupply | causes_onto.ValveIntactUnintentionallyClosed |
                                                  causes_onto.ValveWronglyClosed) &
                      safeguardDependsOnRiskCategory.some(risk_assessment_onto.A | risk_assessment_onto.B)
                      |
                      (safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity) &
                       safeguardPreventsCause.some(causes_onto.MissingImpeller | causes_onto.DeadHeadingOfPump) &
                       safeguardDependsOnRiskCategory.some(risk_assessment_onto.A | risk_assessment_onto.B)))]

class ImplementNoFlowWarning(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity) &
                     safeguardPreventsCause.some(causes_onto.MissingImpeller) &
                     safeguardDependsOnRiskCategory.some(risk_assessment_onto.C | risk_assessment_onto.D)]

class PeriodicalSampleTaking(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsCause.some(causes_onto.ContaminationByWater |
                                                 causes_onto.WrongTankLinedUp |
                                                 causes_onto.InadvertentContamination |
                                                 causes_onto.TooLittleStabilizer |
                                                 causes_onto.TooLittleInhibitor)]

class PressureVacuumReliefValve(Safeguard):
    equivalent_to = [Safeguard &
                     ((safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity) &
                       safeguardPreventsEffect.some(effect_onto.LossOfMechanicalIntegrity))
                      |
                      (safeguardInvolvesEquipmentEntity.some(equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank)) &
                       safeguardPreventsCause.some(causes_onto.InsufficientThermalOutbreathing) &
                       safeguardPreventsEffect.some(effect_onto.Fracture))
                      |
                      (safeguardInvolvesEquipmentEntity.some(equipment_onto.hasApparatus.some(equipment_onto.AtmosphericStorageTank)) &
                       safeguardOfDeviation.some(deviation_onto.HighPressure) &
                       safeguardPreventsEffect.some(effect_onto.Fracture)))]

PressureVacuumReliefValve.comment = ["also known as 'conservation vent valve'"]

class CollectingBasin(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardMitigatesConsequence.some(consequence_onto.LossOfPrimaryContainment) &
                      safeguardInvolvesEquipmentEntity.some(equipment_onto.PressureVesselEntity | equipment_onto.PressureReceiverEntity))]

class ValveLockedClosed(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.BypassOpened)
                      |
                      (safeguardPreventsCause.some(causes_onto.DrainValveInadvertentlyOpened) &
                       safeguardPreventsEffect.some(effect_onto.PoolFormation)))]

class SwingCheckValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.WaterHammer) &
                      safeguardInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity))]

class SurgeReliefValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.WaterHammer) &
                      safeguardInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity |
                                                            equipment_onto.PumpEntity))]
```

Appendix Listing P-1: Class definitions of safeguard concepts Part I

```
class FlareSystem(Safeguard):
    equivalent to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(
                         equipment_onto.hasApparatus.some(equipment_onto.PressureVessel)) &
                     impliesSafeguard.some(PressureReliefValve) &
                     safeguardInvolvesSubstance.some(
                       substance_onto.hasHazardClass.some(
                           substance_onto.FlammableLiquidCategory1 |
                           substance_onto.FlammableLiquidCategory2 |
                           substance_onto.FlammableLiquidCategory3 |
                           substance_onto.FlammableGasCategory1 |
                           substance_onto.FlammableGasCategory2 |
                           substance_onto.PyrophoricGasCategory1 |
                           substance_onto.AerosolCategory1 |
                           substance_onto.AerosolCategory2
                     ))]

class IncreaseClosingTimeOfValve(Safeguard):
    equivalent to = [Safeguard &
                     safeguardPreventsCause.some(causes onto.WaterHammer)]

class PulsationDampener(Safeguard):
    equivalent to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.WaterHammer) &
                      safeguardInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity))]

class ConsiderMaterialSelection(Safeguard):
    equivalent to = [Safeguard &
                     safeguardPreventsEffect.some(effect onto.GenerationOfElectrostaticCharge) |
                     safeguardOfDeviation.some(deviation onto.HighCorrosion)]

class GasBalanceLine(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsCause.some(causes_onto.LiquidTransferWithoutCompensation) &
                     safeguardInvolvesEquipmentEntity.some(
                         equipment onto.hasApparatus.some(equipment onto.AtmosphericStorageTank))]

class IncreaseSafetyIntegrityLevel(Safeguard):
    equivalent to = [Safeguard &
                     safeguardDependsOnRiskCategory.some(risk_assessment_onto.C |
                                                         risk_assessment_onto.B) &
                     safeguardPreventsUnderlyingCause.some(causes_onto.LevelIndicatorControllerFailure)
                     ]

class ThermalProtectionTrip(Safeguard):
    equivalent to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment onto.PumpEntity | equipment onto.CompressorEntity) &
                      (safeguardOfDeviation.some(deviation onto.HighTemperature) |
                       (safeguardPreventsEffect.some(effect onto.Overheating |
                                                     effect_onto.FluidCirculatesInsidePump))
                      |
                       (safeguardOfDeviation.some(deviation_onto.HighTemperature) &
                        safeguardPreventsEffect.some(effect_onto.ExcessiveDischargeTemperature))))
                     ]

class SoftStartStopControl(Safeguard):
    equivalent to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment onto.PumpEntity) &
                      safeguardPreventsCause.some(causes onto.WaterHammer))
                     ]

class InstallHighLevelAlarm(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardDependsOnRiskCategory.some(risk_assessment_onto.C |
                                                         risk assessment onto.D) &
                     safeguardPreventsEffect.some(effect onto.Overfilling)]

class InstallHighTemperatureAlarm(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment_onto.hasApparatus.some(
                                                              equipment_onto.AtmosphericStorageTank |
                                                              equipment_onto.PressureVessel)) &
                      safeguardPreventsEffect.some(effect onto.UnintendedExothermicPolymerization) &
                      safeguardInvolvesSubstance.some(
                         substance onto.hasStabilityReactivityInformation.some(
                             substance onto.PolymerizesExothermicallyWhenExposedToHeat)))]

class ProvideLeakageMonitoring(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity) &
                     safeguardPreventsEffect.some(effect onto.PoolFormation)]

class ProvideAntiCorrosionCoating(Safeguard):
    equivalent to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity) &
                     safeguardOfDeviation.some(deviation onto.HighCorrosion)]

class ImplementLowLevelAlarm(Safeguard):
    equivalent_to = [Safeguard &
                     ((safeguardOfDeviation.some(deviation_onto.LowLevel) &
                       safeguardInvolvesEquipmentEntity.some(
                          equipment onto.hasApparatus.some(equipment onto.PressureVessel |
                                                           equipment onto.AtmosphericStorageTank)) &
                       safeguardDependsOnRiskCategory.some(risk assessment onto.A | risk assessment onto.B))
                      |
                      (safeguardOfDeviation.some(deviation_onto.LowLevel) &
                       safeguardPreventsCause.some(causes_onto.LossOfInflow |
                                                   causes_onto.IncorrectFilling |
                                                   causes_onto.ClosedInletValve) &
                       safeguardDependsOnRiskCategory.some(risk_assessment_onto.A | risk_assessment_onto.B))
                      |
                      (safeguardPreventsEffect.some(effect onto.EmptyingOfContainer) &
                       safeguardDependsOnRiskCategory.some(risk assessment onto.A | risk assessment onto.B)))]

class ImplementLowLevelWarning(Safeguard):
    equivalent_to = [Safeguard &
                     ((safeguardOfDeviation.some(deviation_onto.LowLevel) &
                      safeguardInvolvesEquipmentEntity.some(
                          equipment_onto.hasApparatus.some(equipment_onto.PressureVessel | equipment_onto.AtmosphericStorageTank)) &
                      safeguardDependsOnRiskCategory.some(risk_assessment_onto.C | risk_assessment_onto.D))
                     |
                     (safeguardOfDeviation.some(deviation onto.LowLevel) &
                      safeguardPreventsCause.some(causes onto.LossOfInflow | causes onto.IncorrectFilling | causes onto.ClosedInletValve) &
                      safeguardDependsOnRiskCategory.some(risk assessment onto.C | risk assessment onto.D))
                      |
                      (safeguardPreventsEffect.some(effect_onto.EmptyingOfContainer) &
                       safeguardDependsOnRiskCategory.some(risk_assessment_onto.C | risk_assessment_onto.D)))
                     ]

class ValveLockedOpen(Safeguard):
    equivalent to = [Safeguard &
                     safeguardPreventsCause.some(causes onto.DeadHeadingOfPump | causes onto.ClosedInletValve)]
```

Appendix Listing P-2: Class definitions of safeguard concepts Part II

```
class InstallPhysicalBarrierAroundTheEquipment(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardInvolvesEquipmentEntity.some(equipment onto.StorageTankEntity) &
                     safeguardPreventsCause.some(causes_onto.PhysicalImpact)]

class InstallPressureLimitationValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.IncorrectSetPointControlValve) &
                      safeguardOfDeviation.some(deviation onto.HighPressure))
                     |
                     (safeguardInvolvesEquipmentEntity.some(equipment onto.InertgasBlanketingEntity) &
                      safeguardOfDeviation.some(deviation onto.HighPressure))]

class InstallRestrictiveFlowOrifice(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.ExcessiveInflow) &
                      safeguardInvolvesEquipmentEntity.some(equipment onto.InertgasBlanketingEntity))]

class InstallCheckValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardOfDeviation.some(deviation onto.ReverseFlow)
                      |
                      (safeguardOfDeviation.some(deviation onto.ElsewhereFlow) &
                       safeguardInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity)))]

class QuenchSystem(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect onto.UnintendedExothermicPolymerization) &
                      safeguardInvolvesEquipmentEntity.some(equipment onto.PressureVesselEntity |
                                                            equipment onto.SettlingTankEntity |
                                                            equipment onto.ReactorEntity |
                                                            equipment_onto.ShellTubeHeatExchangerEntity |
                                                            equipment_onto.StorageTankEntity))]

class VacuumProofDesign(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment onto.PressureReceiverEntity |
                                                            equipment onto.SettlingTankEntity |
                                                            equipment onto.PressureVesselEntity) &
                      safeguardOfDeviation.some(deviation_onto.LowPressure) &
                      safeguardPreventsEffect.some(effect_onto.LossOfMechanicalIntegrity) &
                      safeguardDependsOnRiskCategory.some(risk_assessment_onto.A | risk_assessment_onto.B))]

class HighPointVent(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment onto.PumpEntity) &
                      safeguardOfDeviation.some(deviation onto.OtherThanComposition | deviation onto.NoFlow | deviation onto.LowFlow) &
                      safeguardPreventsCause.some(causes onto.EntrainedAir))]

class CheckIfFreeBlowOffPossible(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity) &
                      safeguardOfDeviation.some(deviation_onto.OtherThanComposition | deviation_onto.NoFlow | deviation_onto.LowFlow) &
                      safeguardPreventsCause.some(causes_onto.EntrainedAir))]

CheckIfFreeBlowOffPossible.comment = ["Depends directly on HighPointVent"]

class UseSealingCaps(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsCause.some(causes_onto.ContaminationInUnloadingLines)]

class DetermineSafetyRelatedOperatingInstructions(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect_onto.PotentialViolentReactionWithOxidizers)
                      |
                      safeguardPreventsUnderlyingCause.some(causes onto.HoseIncorrectlyConnected |
                                                            causes onto.IncorrectSetPointControlValve |
                                                            causes onto.IncorrectPressureAdjustment))]

class EmergencyPressureReliefValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect_onto.Fracture) &
                      safeguardPreventsUnderlyingCause.some(causes onto.ExternalFire) &
                      safeguardInvolvesEquipmentEntity.some(equipment onto.hasApparatus.some(
                                                                  equipment onto.AtmosphericStorageTank |
                                                                  equipment onto.OpenVessel)))]

EmergencyPressureReliefValve = ["Located on low pressure storage tanks"]

class AddStabilizer(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsEffect.some(effect onto.UnintendedExothermicPolymerization)]

class FailsafeFeedStop(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsEffect.some(effect onto.InsufficientInertization)]

class FlameArrestor(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity |
                                                            equipment onto.SettlingTankEntity) &
                      safeguardMitigatesConsequence.some(consequence onto.RiskOfExplosiveAtmosphere) &
                      safeguardInvolvesSubstance.some(substance onto.hasHazardClass.some(
                                                            substance onto.FlammableLiquidCategory1 |
                                                            substance onto.FlammableLiquidCategory2 |
                                                            substance_onto.FlammableLiquidCategory3)))]

FlameArrestor.comment = [
    "methanol tank, should be protected with flame arrestor and nitrogen blanketing (more than 3000 m³, API2000)"]

class RegularPlantInspection(Safeguard):
    equivalent_to = [Safeguard &
                     ((safeguardPreventsCause.some(causes onto.ExternalLeakage) | safeguardPreventsEffect.some(effect onto.PoolFormation))
                      |
                      safeguardMitigatesConsequence.some(consequence_onto.LossOfPrimaryContainment))
                     ]

class ElaborationOfMaintenanceConcept(Safeguard):
    equivalent_to = [Safeguard &
                     ((safeguardPreventsCause.some(causes_onto.PumpSealFailure) & safeguardPreventsEffect.some(effect_onto.PoolFormation))
                     |
                     (safeguardPreventsUnderlyingCause.some(causes onto.LossOfLeakTightness | causes onto.BrokenHose) &
                      safeguardPreventsCause.some(causes onto.ExternalLeakage))
                     |
                     (safeguardPreventsUnderlyingCause.some(causes_onto.ControlValveFailsClosed |
                                                            causes_onto.ControlValveFailsOpen) &
                      safeguardDependsOnRiskCategory.some(risk_assessment_onto.C | risk_assessment_onto.D))
                     )]

class LimitationOfTheFlowVelocity(Safeguard):
    equivalent_to = [Safeguard &
                     safeguardPreventsEffect.some(effect onto.GenerationOfElectrostaticCharge)
                     ]
```

Appendix Listing P-3: Class definitions of safeguard concepts Part III

```
class ConsiderMinimumTankDistance(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity) &
                      safeguardMitigatesConsequence.some(consequence_onto.LossOfPrimaryContainment) &
                      safeguardPreventsEffect.some(effect_onto.LossOfMechanicalIntegrity | effect_onto.Fracture) &
                      safeguardInvolvesSubstance.some(substance_onto.hasFlashpointInKelvin <= 273.15 + 55.0))]

ConsiderMinimumTankDistance.comment = ["TRGS 509, pp. 37, <= 55 °C"]

class CheckPressureClassPiping(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect_onto.Fracture) &
                      safeguardInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity | equipment_onto.ValveEntity) &
                      safeguardOfDeviation.some(deviation_onto.HighPressure))]

class UseNormallyOpenValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.ValveWronglyClosed) & safeguardPreventsEffect.some(effect_onto.EmptyingOfContainer))]

class AvoidingBlockedLiquids(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardInvolvesEquipmentEntity.some(equipment_onto.ConnectionPipeEntity) &
                      safeguardPreventsCause.some(causes_onto.AbnormalHeatInput))]

class InstallHeatTracingSystem(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect_onto.DrainlineFracture) &
                      safeguardPreventsCause.some(causes_onto.ContaminationByWaterAndTemperatureFallsBelowFreezingPoint))]

class InstallThermalExpansionReliefValve(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsEffect.some(effect_onto.PressureExceedingDesignPressure) &
                      safeguardPreventsCause.some(causes_onto.ThermalExpansion) &
                      safeguardPreventsUnderlyingCause.some(causes_onto.BlockedPipingAndHeatInput))]

class StandardOperationProcedure(Safeguard):
    equivalent_to = [Safeguard &
                     (safeguardPreventsCause.some(causes_onto.DrainValveInadvertentlyOpened | causes_onto.OperationBelowMinimumFlowRate)
                     |
                      (safeguardPreventsUnderlyingCause.some(causes_onto.OperationalError) &
                       safeguardDependsOnRiskCategory.some(risk_assessment_onto.C | risk_assessment_onto.D))
                     |
                      (safeguardPreventsUnderlyingCause.some(causes_onto.MaintenanceError) &
                       safeguardDependsOnRiskCategory.some(risk_assessment_onto.C | risk_assessment_onto.D)))]

class ProvideGroundingOfPlant(Safeguard):
    equivalent_to = [Safeguard & safeguardPreventsEffect.some(effect_onto.GenerationOfElectrostaticCharge)]

class PeriodicInspection(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard & safeguard_onto.safeguardPreventsCause.some(MalfunctionLubricationSystem)]

class ImprovedOilSeparation(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard & safeguard_onto.safeguardPreventsEffect.some(IncreasedOilDischarge)]

class QuickActionStopValve(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.AirCooledCondenserEntity) &
                     safeguard_onto.safeguardOfDeviation.some(deviation_onto.ElsewhereFlow)
                     ]

class PurgerUnit(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     safeguard_onto.safeguardPreventsCause.some(causes_onto.NonCondensables)]

class OverflowValve(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     ((safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity |
                                                            equipment_onto.hasMaterialTransferEquipment.some(equipment_onto.CentrifugalPump |
                                                                                                             equipment_onto.ReciprocatingPump)))
                      & safeguard_onto.safeguardPreventsEffect.some(effect_onto.FluidCirculatesInsidePump) &
                      safeguard_onto.safeguardPreventsCause.some(DeadHeadingOfPump | causes_onto.ValveIntactUnintentionallyClosed))]

class MinimumFlowProtectionSystem(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardPreventsCause.some(OperationBelowMinimumFlowRate) &
                      safeguard_onto.safeguardDependsOnRiskCategory.some(risk_assessment_onto.A | risk_assessment_onto.B) &
                      safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity))]

MinimumFlowProtectionSystem.comment = ["e.g. continues bypass, automatic recirculation valve"]

class LowFlowProtectionTrip(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardPreventsCause.some(OperationBelowMinimumFlowRate) &
                      safeguard_onto.safeguardOfDeviation.some(deviation_onto.LowFlow))]

class DryRunProtection(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity) &
                      safeguard_onto.safeguardPreventsEffect.some(PumpRunningDry))]

class InstallPumpInducer(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardPreventsEffect.some(Cavitation) &
                      safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity))]

class IncreaseSuctionPressure(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardPreventsEffect.some(Cavitation) &
                      safeguard_onto.safeguardPreventsCause.some(InsufficientNPSH) &
                      safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.PumpEntity))]

class DefineMaximumFillLevel(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity) &
                      safeguard_onto.safeguardPreventsEffect.some(effect_onto.Overfilling))]

class TemperatureControllerHighAlarm(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardPreventsCause.some(causes_onto.AbnormallyHotIntake | CoolingFailure) &
                      safeguard_onto.safeguardOfDeviation.some(deviation_onto.HighTemperature))]

class EmergencyCooling(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardPreventsCause.some(CoolingFailure) &
                      safeguard_onto.safeguardOfDeviation.some(deviation_onto.HighTemperature))]

class EmergencyStabilization(safeguard_onto.Safeguard):
    equivalent_to = [safeguard_onto.Safeguard &
                     (safeguard_onto.safeguardInvolvesEquipmentEntity.some(equipment_onto.StorageTankEntity |
                                                                           equipment_onto.SettlingTankEntity |
                                                                           equipment_onto.ReactorEntity |
                                                                           equipment_onto.PressureVesselEntity |
                                                                           equipment_onto.PressureReceiverEntity) &
                      safeguard_onto.safeguardPreventsEffect.some(effect_onto.UnintendedExothermicPolymerization))]
```

Appendix Listing P-4: Class definitions of safeguard concepts Part IV

Appendix Q Inference algorithm

```
def equipment based hazard specific deviation(deviation, args):

    # === Input
    process unit = args[0]
    substance = args[1]
    environment = args[2]
    subsequent deviation loop = args[3]
    subsequent deviations = args[4]
    stack elements = args[5]

    # === Loop over causes ========================================================================================
    preliminary scenario list = []
    extended_scenarion_list = []
    cause_list = None
    cause = None
    likelihood_ = None
    underlying cause  = None

    if not subsequent deviation loop:
        # = Cause =================================================================================================
        # =========================================================================================================
        deviation1, deviation2 = assemble_deviation(deviation)

        cause = causes_onto.Cause(isCauseOfDeviation=deviation1,
                                  causeInvolvesSecondDeviation=deviation2,
                                  causeInvolvesEquipmentEntity=[process_unit.onto_object],
                                  causeRequiresBoundaryCondition=process unit.boundary condition,
                                  causeInvolvesSiteInformation=[environment.onto object],
                                  causeInvolvesSubstance=[substance.onto object])  # functional, nur 1 substanz übergeben

        sync_reasoner(debug=0)

    else:
        if isinstance(deviation, dict):
            if isinstance(deviation[prep.DictName.subsequent_deviation], str):
                cause_list = deviation[prep.DictName.explanation]
                underlying cause  = deviation[prep.DictName.underlying cause]()
                likelihood  = deviation[prep.DictName.likelihood]()
                deviation = [upper onto.pse onto[deviation[prep.DictName.subsequent deviation]]()]
            else:
                cause_list = deviation[prep.DictName.explanation]
                underlying_cause_ = deviation[prep.DictName.underlying_cause]()
                likelihood_ = deviation[prep.DictName.likelihood]()
                deviation = deviation[prep.DictName.subsequent_deviation]()

    if cause and cause.is a:
        cause lst = prep.get inferred results(cause)

        for idx, cause  in enumerate(cause lst):
            underlying_causes, effect, consequence, = assemble_concept_instance(substance,
                                                                                process_unit,
                                                                                deviation,
                                                                                [],
                                                                                cause ,
                                                                                False)
            # if likelihood cause and likelihood overall:
            scenario = {prep.DictName.cause: cause (),
                        prep.DictName.underlying cause: underlying causes,
                        prep.DictName.effect: effect,
                        prep.DictName.consequence: consequence}

            preliminary scenario list.append(scenario)

    elif cause list:
        underlying causes, effect, consequence, = assemble concept instance(substance,
                                                                            process unit,
                                                                            deviation,
                                                                            underlying cause ,
                                                                            [],
                                                                            True)

        scenario = {prep.DictName.cause: cause list,
                    prep.DictName.underlying cause: underlying causes,
                    prep.DictName.effect: effect,
                    prep.DictName.consequence: consequence}

        preliminary_scenario_list.append(scenario)
    else:
        underlying_causes, effect, consequence = assemble_concept_instance(substance,
                                                                           process_unit,
                                                                           deviation,
                                                                           [],
                                                                           [],
                                                                           False)

        scenario = {prep.DictName.cause: [],
                    prep.DictName.underlying_cause: underlying_causes,
                    prep.DictName.effect: effect,
                    prep.DictName.consequence: consequence,
                    }

        preliminary scenario list.append(scenario)

    sync reasoner(debug=0)


infer_follow_up(process_unit,
                substance,
                deviation,
                environment,
                subsequent deviation loop,
                likelihood ,
                underlying cause ,
                stack elements,
                preliminary_scenario_list,
                extended_scenarion_list)
```

Appendix Listing Q-1: Python code of the main routine of the equipment-based mode

```
def infer follow up(process unit,
                    substance,
                    deviation,
                    environment,
                    subsequent_deviation_loop,
                    likelihood_,
                    underlying_cause_,
                    stack_elements,
                    preliminary scenario list,
                    extended scenarion list):

    underlying cause list = []
    for scenario in preliminary scenario list:
        # Hint: This identifies more u-causes
        underlying_cause_list = prep.get_inferred_results(scenario[prep.DictName.underlying_cause])
        # Hint: than this
        # underlying cause list = scenario[prep.DictName.underlying cause].is instance of

        # Remove non relevant components of the list
        if upper onto.owl.Thing in underlying cause list:
            underlying cause list.remove(upper onto.owl.Thing)
        if upper onto.pse onto.UnderlyingCause in underlying cause list:
            underlying cause list.remove(upper onto.pse onto.UnderlyingCause)

        # === Creation of a separate scenario for each underlying_cause
        if len(underlying cause list) >= 1:
            for underlying cause in underlying cause list:
                for consequence in scenario[prep.DictName.consequence].is instance of:
                    for effect in scenario[prep.DictName.effect].is instance of:
                        extended scenarion list.append({prep.DictName.cause: scenario[prep.DictName.cause],
                                                        prep.DictName.underlying cause: underlying cause(),
                                                        prep.DictName.effect: effect(),
                                                        prep.DictName.consequence: consequence()})
        else:
            for consequence in scenario[prep.DictName.consequence].is_instance_of:
                for effect in scenario[prep.DictName.effect].is instance of:
                    extended scenarion list.append({prep.DictName.cause: scenario[prep.DictName.cause],
                                                    prep.DictName.effect: effect(),
                                                    prep.DictName.underlying cause: scenario[prep.DictName.underlying cause],
                                                    prep.DictName.consequence: consequence()})

    scenario_list = extended_scenarion_list

    for scenario in scenario_list:
        deviation1, deviation2 = assemble_deviation(deviation)
        if not subsequent deviation loop:
            scenario[prep.DictName.likelihood cause] = risk assessment onto.Likelihood(
                                        likelihoodInvolvesUnderlyingcause=[scenario[prep.DictName.underlying cause]],
                                        likelihoodInvolvesCause=[scenario[prep.DictName.cause]],
                                        likelihoodInvolvesEquipment=[process_unit.onto_object],
                                        likelihoodInvolvesSiteInformation=[environment.onto_object],
                                        likelihoodInvolvesDeviation=deviation1,
                                        likelihoodRequiresBoundaryCondition=process_unit.boundary_condition)

            scenario[prep.DictName.likelihood_effect] = risk_assessment_onto.Likelihood(
                                                likelihoodInvolvesEffect=[scenario[prep.DictName.effect]],
                                                likelihoodInvolvesCause=[scenario[prep.DictName.cause]],
                                                likelihoodInvolvesEquipment=[process unit.onto object],
                                                likelihoodInvolvesSiteInformation=[environment.onto object],
                                                likelihoodInvolvesDeviation=deviation1,
                                                likelihoodRequiresBoundaryCondition=process_unit.boundary_condition)

        else:
            if likelihood_:
                scenario[prep.DictName.likelihood cause] = likelihood
            else:
                scenario[prep.DictName.likelihood cause] = risk assessment onto.Likelihood(
                    likelihoodInvolvesUnderlyingcause=underlying cause ,
                    likelihoodInvolvesEquipment=[process_unit.onto_object],
                    likelihoodInvolvesDeviation=deviation1,
                    likelihoodRequiresBoundaryCondition=process_unit.boundary_condition)

            if underlying cause :
                scenario[prep.DictName.underlying cause] = underlying cause

            scenario[prep.DictName.likelihood effect] = risk assessment onto.Likelihood(
                likelihoodInvolvesEffect=[scenario[prep.DictName.effect]],
                likelihoodInvolvesEquipment=[process unit.onto object],
                likelihoodInvolvesDeviation=deviation1,
                likelihoodRequiresBoundaryCondition=process_unit.boundary_condition)

        # Fix for excluding delayed ignition in case of external fire, makes not sense
        if scenario[prep.DictName.underlying cause]:
            boundary conditions  = process unit.boundary condition.copy()
            if isinstance(scenario[prep.DictName.underlying cause], list):
                pass
            else:
                if scenario[prep.DictName.underlying_cause].is_a[0] == causes_onto.ExternalFire:
                    for bc in boundary_conditions_:
                        if bc.is_a[0] == boundary_onto.DelayedIgnition:
                            boundary conditions .remove(bc)
        else:
            boundary conditions  = process unit.boundary condition.copy()
```

Appendix Listing Q-2: Python code of the “follow up” function of the equipment-based mode I

```
    scenario[prep.DictName.consequence 2nd] = consequence onto.Consequence(
        consequenceInvolvesSubstance=[substance.onto object],
        consequenceInvolvesEquipmentEntity=[process unit.onto object],
        isConsequenceOfDeviation=deviation1,
        consequenceImpliedByUnderlyingcause=[scenario[prep.DictName.underlying_cause]],
        isSubsequentConsequence=[scenario[prep.DictName.consequence]],
        consequenceRequiresBoundaryCondition=boundary_conditions_)

    scenario[prep.DictName.consequence 3rd] = consequence onto.Consequence(
        consequenceInvolvesSubstance=[substance.onto object],
        consequenceInvolvesEquipmentEntity=[process unit.onto object],
        isConsequenceOfDeviation=deviation1,
        consequenceImpliedByUnderlyingcause=[scenario[prep.DictName.underlying cause]],
        isSubsequentConsequence=[scenario[prep.DictName.consequence_2nd]],
        consequenceRequiresBoundaryCondition=boundary_conditions_)

    scenario[prep.DictName.severity] = risk assessment onto.SeverityCategory(isSeverityOfConsequence=[scenario[prep.DictName.consequence],
                                                                                                       scenario[
                                                                                                           prep.DictName.consequence 2nd],
                                                                                                       scenario[
                                                                                                           prep.DictName.consequence 3rd]],
                                                                             isSeverityOfEffect=[scenario[prep.DictName.effect]],
                                                                             severityInvolvesSubstance=[substance.onto object],
                                                                             severityInvolvesEquipment=[
                                                                                 process_unit.onto_object],
                                                                             severityRequiresBoundaryCondition=process unit.boundary condition)

    scenario[prep.DictName.risk cause] = risk assessment onto.RiskCategory(involvesSeverity=[scenario[prep.DictName.severity]],
                                                                           involvesLikelihood=[scenario[prep.DictName.likelihood cause]])

    scenario[prep.DictName.risk effect] = risk assessment onto.RiskCategory(involvesSeverity=[scenario[prep.DictName.severity]],
                                                                            involvesLikelihood=[scenario[prep.DictName.likelihood_effect]])

    if not isinstance(scenario[prep.DictName.cause], str) and not isinstance(scenario[prep.DictName.cause], list):
        cause_argument = [scenario[prep.DictName.cause]]
    else:
        cause argument = []

    scenario[prep.DictName.safeguard] = safeguard onto.Safeguard(safeguardOfDeviation=deviation1,
                                                                 safeguardPreventsEffect=[scenario[prep.DictName.effect]],
                                                                 safeguardPreventsCause=cause_argument,
                                                                 safeguardPreventsUnderlyingCause=[scenario[prep.DictName.underlying_cause]],
                                                                 safeguardMitigatesConsequence=[scenario[prep.DictName.consequence],
                                                                                                scenario[prep.DictName.consequence_2nd],
                                                                                                scenario[prep.DictName.consequence_3rd]
                                                                                                ],
                                                                 safeguardDependsOnRiskCategory=[scenario[prep.DictName.risk cause],
                                                                                                 scenario[prep.DictName.risk effect]],
                                                                 safeguardInvolvesEquipmentEntity=[process unit.onto object],
                                                                 safeguardInvolvesSubstance=[substance.onto_object])

    scenario[prep.DictName.safeguard_2nd] = safeguard_onto.Safeguard(
        impliesSafeguard=[scenario[prep.DictName.safeguard]],
        safeguardDependsOnRiskCategory=[scenario[prep.DictName.risk_cause], scenario[prep.DictName.risk_effect]],
        safeguardInvolvesEquipmentEntity=[process_unit.onto_object],
        safeguardInvolvesSubstance=[substance.onto object])

sync reasoner(debug=0)

# === Pass results
for scenario in scenario_list:
    consequence_list = [scenario[prep.DictName.consequence], scenario[prep.DictName.consequence_2nd], scenario[prep.DictName.consequence_3rd]]

    # === Prioritize identified risk effect-based since likelihood of it is based on specific generic data
    if scenario[prep.DictName.risk effect].is a[0].name == "A" or \
            scenario[prep.DictName.risk effect].is a[0].name == "B" or \
            scenario[prep.DictName.risk effect].is a[0].name == "C" or \
            scenario[prep.DictName.risk effect].is a[0].name == "D":
        scenario[prep.DictName.risk] = scenario[prep.DictName.risk_effect]
        scenario[prep.DictName.likelihood] = scenario[prep.DictName.likelihood_effect]
    else:
        scenario[prep.DictName.risk] = scenario[prep.DictName.risk_cause]

        # == Catch special case that in propagation mode no risk is determined but likelihood is present
        if scenario[prep.DictName.likelihood effect].is a[0] != upper onto.pse onto.Likelihood:
            scenario[prep.DictName.likelihood] = scenario[prep.DictName.likelihood effect]
        else:
            scenario[prep.DictName.likelihood] = scenario[prep.DictName.likelihood cause]

    propagated_consequence_found = False
    actual consequence found = False

    # === Identify propagating and actual results
    actual consequences = []
    for consequence  in consequence list:
        if consequence .is a[0] == consequence onto.PROPAGATED CONSEQUENCE:
            propagated consequence found = True
            break
        elif consequence_.is_a[0] != consequence_onto.Consequence:
            actual_consequence_found = True
            actual consequences.append(consequence )

    if actual consequence found:
        prep.log scenario(0,
                          process unit,
                          substance.name,
                          deviation,
                          scenario[prep.DictName.cause],
                          scenario[prep.DictName.effect],
                          scenario[prep.DictName.underlying_cause],
                          actual consequences,
                          [scenario[prep.DictName.safeguard], scenario[prep.DictName.safeguard 2nd]],
                          False,  # propagated
                          scenario[prep.DictName.risk])

    if propagated_consequence_found and not actual_consequence_found:
        data = {prep.DictName.equipment: process_unit,
                prep.DictName.substance: substance,
                prep.DictName.deviation: deviation,
                prep.DictName.cause: scenario[prep.DictName.cause],
                prep.DictName.effect: scenario[prep.DictName.effect],
                prep.DictName.underlying cause: scenario[prep.DictName.underlying cause],
                prep.DictName.likelihood: scenario[prep.DictName.likelihood],
                prep.DictName.consequence: consequence list,
                prep.DictName.safeguard: [scenario[prep.DictName.safeguard], scenario[prep.DictName.safeguard_2nd]],
                prep.DictName.id: process_unit.identifier}

        # Catch duplicate in relevant propagation stack
        if isinstance(data[prep.DictName.cause], str):
            if not any(d[prep.DictName.cause] == data[prep.DictName.cause] and
                       d[prep.DictName.underlying cause].is a[0].name == data[prep.DictName.underlying cause].is a[0].name for d in stack elements):
                stack elements.append(data)
        else:
            if not any(d[prep.DictName.cause].is_a[0].name == data[prep.DictName.cause].is_a[0].name and
                       d[prep.DictName.underlying_cause].is_a[0].name == data[prep.DictName.underlying_cause].is_a[0].name for d in stack_elements):
                stack_elements.append(data)
```

Appendix Listing Q-3: Python code of the “follow up” function of the equipment-based mode II

```
# === Identify subsequent deviations based on the results
if not subsequent deviation loop:
    if subsequent deviation loop:
        current_deviation = deviation[prep.DictName.subsequent_deviation].copy()
    else:
        if isinstance(deviation, list):
            if len(deviation) > 1:
                current_deviation = []
                for dev in deviation:
                    current deviation.append(dev.is a[0])
            else:
                current deviation = deviation[0].is a[0]
        else:
            current_deviation = deviation.is_a[0]

    # === check existence of apparatus
    if process unit.apparatus:
        apparatus = process unit.apparatus[0].is a
        if isinstance(apparatus, list):
            apparatus = apparatus[0]
    else:
        apparatus = "None"

    for scenario in extended_scenarion_list:
        # === Infer subsequent deviations based on effect
        if scenario[prep.DictName.effect]:
            for effect in scenario[prep.DictName.effect].is a:
                current case = {cbr.CaseAttributes.EquipmentEntity: process unit.onto object.is a[0],
                                cbr.CaseAttributes.Event: effect,
                                cbr.CaseAttributes.Apparatus: apparatus,
                                cbr.CaseAttributes.IntendedFunction: process unit.intended function[0].is a[0],
                                cbr.CaseAttributes.SubstancePhase:
                                    substance.onto_object.hasStateOfAggregation[0].is_a[0]}

                match = cbr.match_case_with_cb(current_case, cbr.propagation_case_base)

                if match:
                    for m in match:
                        if match != current deviation:
                            dev = {}
                            fmt_phase = str(substance.onto_object.hasStateOfAggregation[0].is_a[0]).split('.', 1)[1]
                            dev[prep.DictName.subsequent_deviation] = m
                            dev[prep.DictName.cause] = scenario[prep.DictName.cause].is_a[0]
                            dev[prep.DictName.underlying_cause] = scenario[prep.DictName.underlying_cause].is_a[0]
                            dev[prep.DictName.explanation] = process_unit.identifier + ": " + dev[
                                prep.DictName.cause].name + " -> " + effect.name + " -> " + dev[
                                prep.DictName.subsequent deviation].name + " " + "(" + fmt phase + ")"
                            dev[prep.DictName.likelihood] = scenario[prep.DictName.likelihood].is a[0]
                            subsequent deviations.append(dev)

        # === Infer subsequent deviations based on cause
        if scenario[prep.DictName.cause]:
            for cause in scenario[prep.DictName.cause].is_a:
                current_case = {cbr.CaseAttributes.EquipmentEntity: process_unit.onto_object.is_a[0],
                                cbr.CaseAttributes.Event: cause,
                                cbr.CaseAttributes.Apparatus: apparatus,
                                cbr.CaseAttributes.IntendedFunction: process unit.intended function[0].is a[0],
                                cbr.CaseAttributes.SubstancePhase:
                                    substance.onto object.hasStateOfAggregation[0].is a[0]}

                match = cbr.match_case_with_cb(current_case, cbr.propagation_case_base)

                if match:
                    for m in match:
                        if match != current deviation:
                            dev = {}
                            fmt phase = \
                                str(substance.onto object.hasStateOfAggregation[0].is a[0]).split('.', 1)[1]
                            dev[prep.DictName.subsequent_deviation] = m
                            dev[prep.DictName.initiating_event] = cause
                            dev[prep.DictName.underlying_cause] = scenario[prep.DictName.underlying_cause].is_a[0]
                            dev[prep.DictName.likelihood] = scenario[prep.DictName.likelihood].is_a[0]
                            dev[prep.DictName.explanation] = process unit.identifier + ": " + cause.name + " -> " + dev[
                                prep.DictName.subsequent deviation].name + " " + "(" + fmt phase + ")"
                            subsequent deviations.append(dev)

    # === Infer dev -> dev
    if config.CONSIDER DEV 2 DEV PROPAGATION:
        if process_unit.intended_function:
            intended_function = process_unit.intended_function[0].is_a[0]
        else:
            intended function = None

        current case = {cbr.CaseAttributes.EquipmentEntity: process unit.onto object.is a[0],
                        cbr.CaseAttributes.Event: current deviation,
                        cbr.CaseAttributes.Apparatus: apparatus,
                        cbr.CaseAttributes.IntendedFunction: intended function,
                        cbr.CaseAttributes.SubstancePhase: substance.onto_object.hasStateOfAggregation[0].is_a[0]}

        match = cbr.match_case_with_cb(current_case, cbr.propagation_case_base)

        if match and match != current deviation:
            fmt phase = str(substance.onto object.hasStateOfAggregation[0].is a[0]).split('.', 1)[1]
            for m in match:
                dev = {prep.DictName.subsequent deviation: m,
                       prep.DictName.explanation: process_unit.identifier + ": " + current_deviation.name + " -> " + m.name + " " + "(" + fmt_phase +
")",
                       prep.DictName.likelihood: scenario[prep.DictName.likelihood].is_a[0]}  # cannot really be estimated
                subsequent_deviations.append(dev)
```

Appendix Listing Q-4: Python code of the routine to identify subsequent deviations

```
def propagation_based_analysis(plant_graph, order, propagation_stacks):
    # create copy of  original plant layout and remove edges that are not in 'order' list
    # therefore a graph is created according to order
    graph = plant_graph.copy()
    for node in plant_graph.nodes(data=True):
        if node[0] not in order:
            graph.remove_node(node[0])

    # === Attach a placeholder
    for stack in propagation_stacks:
        stack[prep.DictName.passed_scenarios] = []

    # Check for cycle since topological sort not working for cycles
    cycle = list(nx.simple_cycles(graph))

    # Consider direction of graph and rearrange order
    if not cycle:
        rel_order = list(nx.topological_sort(graph))  # TODO: unit test, if this is working
    else:
        rel_order = list(graph.nodes)

    previous_case = {}
    consumed_flag = False
    print("================== PROPAGATION MODE ==================")
    # for index in range(len(plant_graph.nodes)):
    for index, node_pos in enumerate(rel_order):
        process_unit = plant_graph.nodes[node_pos]["data"]
        # Do not assume propagation in case the sink has been reached
        if process_unit == equipment_onto.SinkEntity:
            break
        substances = plant_graph.nodes[node_pos]["substances"]
        # === Loop over substances
        for s, substance in enumerate(substances):
            print("========= Substance: {} = {} of {} =========".format(substance.name, s + 1, len(substances)))
            for m, mode in enumerate(process_unit.operating_mode):
                if mode:
                    print("========= Mode of Operation: {} = {} of {} =========".format(mode.name, m + 1, len(process_unit.operating_mode)))
                # === if no mode is specified continue
                if not mode:
                    continue
                else:
                    process_unit.onto_object.hasOperationMode = [mode()]

                considered_deviations = []

                last_element = len(rel_order) - 1
                no_passed_scenario_flag = True

                if 1 <= index < last_element:
                    pos = rel_order[index - 1]
                    # === Remove duplicates
                    scenario_list = [i for n, i in enumerate(propagation_stacks[pos][prep.DictName.scenario]) if i not in propagation_stacks[pos][prep.DictName.scenario][n
+ 1:]]
                    # === Set flag
                    no_passed_scenario_flag = False
                    consumed_flag = False
                    # === Loop over propagating scenarios
                    for scenario in scenario_list:
                        # === Create input object from scenario, TODO: use scenario object directly
                        subsequent_deviation = assemble_input_object(scenario)
                        # === infer scenarios
                        propagation_based_hazard(subsequent_deviation,
                                                 process_unit,
                                                 substance,
                                                 False,
                                                 previous_case,
                                                 consumed_flag)
                        # === Propagate partial scenario in case no effect have been identified
                        if not consumed_flag:
                            considered_deviations.append(subsequent_deviation)

                    # === consider passed propagation that had no effect in previous equipment
                    if propagation_stacks[pos][prep.DictName.passed_scenarios]:
                        consumed_flag = False
                        for propagation in propagation_stacks[pos][prep.DictName.passed_scenarios]:
                            propagation_based_hazard(propagation,
                                                     process_unit,
                                                     substance,
                                                     False,  # last equipment in graph
                                                     previous_case,
                                                     consumed_flag)
                            # === Propagate partial scenario in case no effect have been identified
                            if not consumed_flag:
                                propagation_stacks[pos + 1][prep.DictName.passed_scenarios].append(propagation)

                    # === Propagate not consumed propagations into next element
                    for propagation in considered_deviations:
                        # Create a list of all values in list of dictionaries
                        list_of_all_values = [value for elem in propagation_stacks[pos + 1][prep.DictName.passed_scenarios] for value in elem.values()]

                        # Catch duplicate in relevant propagation stack
                        if propagation[prep.DictName.explanation] not in list_of_all_values and propagation[prep.DictName.underlying_cause] not in list_of_all_values:
                            propagation_stacks[pos + 1][prep.DictName.passed_scenarios].append(propagation)

                elif index == len(rel_order) - 1:
                    pos = rel_order[index - 1]
                    if propagation_stacks[pos][prep.DictName.passed_scenarios]:
                        propagation_stacks[pos][prep.DictName.passed_scenarios] = [i for n, i in enumerate(
                            propagation_stacks[pos][prep.DictName.passed_scenarios]) if i not in propagation_stacks[pos][
                                                                                       prep.DictName.passed_scenarios][n + 1:]]

                        for propagation in propagation_stacks[pos][prep.DictName.passed_scenarios]:
                            propagation_based_hazard(propagation,
                                                     process_unit,
                                                     substance,
                                                     True,  # last equipment in graph
                                                     previous_case,
                                                     consumed_flag)
                    #  Reached in case no scenarios are passed but the end is already reached
                    if no_passed_scenario_flag:
                        # === Remove duplicates
                        scenario_list = [i for n, i in enumerate(propagation_stacks[pos][prep.DictName.scenario]) if
                                         i not in propagation_stacks[pos][prep.DictName.scenario][n + 1:]]

                        for p, propagation in enumerate(scenario_list):
                            print("=== LAST NODE: {} of {} scenarios ===".format(p, len(scenario_list)))
                            # === is case there is an effect means that no consequence was found for the equipment previously
                            if propagation[prep.DictName.effect].is_a[0] != effect_onto.Effect:
                                complete_propagated_scenario(propagation, process_unit, substance)
                            else:
                                # === Create input object from scenario,
                                propagation = assemble_input_object(propagation)
                                # == infer scenarios
                                if propagation:
                                    propagation_based_hazard(propagation,
                                                             process_unit,
                                                             substance,
                                                             True,  # last equipment in graph
                                                             previous_case,
                                                             consumed_flag)
```

Appendix Listing Q-5: Implemented algorithm of the propagation-based analysis

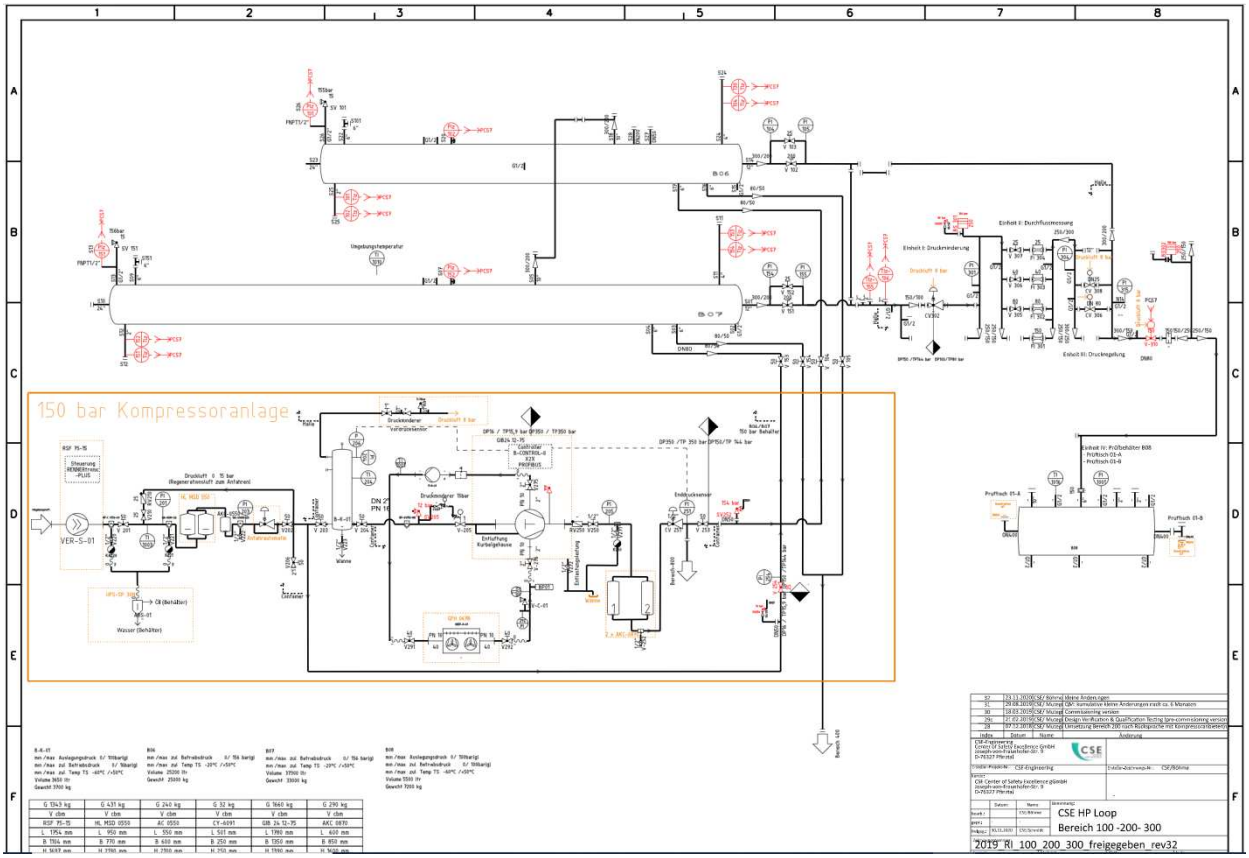
```
def propagation based hazard(devex, process unit, substance, last equipment entity, previous case, consumed flag):
    global results
    try:
        if isinstance(devex[prep.DictName.initiating_event].is_a[0], ThingClass):
            cause = [devex[prep.DictName.initiating_event]]
        else:
            cause = []
    except:
        cause = []

    preliminary scenario list = []
    scenario list = []
    inferred effects = []
    safeguard = None

    cause_list = [devex[prep.DictName.explanation]]
    deviation = devex[prep.DictName.subsequent deviation]
    underlying cause = devex[prep.DictName.underlying cause]

    # catch the case where by mistake no safeguard is provided
    if not devex[prep.DictName.safeguard]:
        devex[prep.DictName.safeguard] = []

    if cause_list:
        underlying_causes, effect, consequence = assemble_concept_instance(substance,
                                                                           process unit,
                                                                           deviation,
                                                                           underlying cause,
                                                                           cause,
                                                                           True)

        if devex[prep.DictName.likelihood]:
            likelihood = devex[prep.DictName.likelihood]
        else:
            likelihood = []

        scenario = {prep.DictName.cause: cause list,
                    prep.DictName.underlying cause: underlying causes,
                    prep.DictName.effect: effect,
                    prep.DictName.consequence: consequence,
                    prep.DictName.likelihood: likelihood
                    }

        inferred_effects.append(effect)
        preliminary_scenario_list.append(scenario)

    sync reasoner(debug=0)

    for scenario in preliminary scenario list:
        for effect in scenario[prep.DictName.effect].is_a:
            if effect.is_a != [owl.Thing]:
                for consequence in scenario[prep.DictName.consequence].is_a:
                    # === inspecting individual
                    for prop in scenario[prep.DictName.consequence].get_properties():
                        if prop == consequence_onto.isConsequenceOfEffect:
                            for value in prop[scenario[prep.DictName.consequence]]:
                                for effect  in value.is a:
                                    # === Effect of consequence matches overall effect
                                    if effect  == effect and scenario[prep.DictName.cause] and effect:
                                        scenario_list.append({prep.DictName.cause: scenario[prep.DictName.cause],
                                                              prep.DictName.underlying_cause: scenario[prep.DictName.underlying_cause],
                                                              prep.DictName.effect: effect(),
                                                              prep.DictName.consequence: consequence(),
                                                              prep.DictName.likelihood: scenario[prep.DictName.likelihood]})
                                        # Consume deviation in case full scenario is found
                                        consumed flag = True

    for scenario in scenario list:
        scenario[prep.DictName.consequence_2nd] = consequence_onto.Consequence(
            consequenceInvolvesSubstance=[substance.onto_object],
            consequenceInvolvesEquipmentEntity=[process_unit.onto_object],
            isConsequenceOfDeviation=deviation,
            isSubsequentConsequence=[scenario[prep.DictName.consequence]],
            consequenceRequiresBoundaryCondition=process unit.boundary condition)

        scenario[prep.DictName.consequence 3rd] = consequence onto.Consequence(
            consequenceInvolvesSubstance=[substance.onto object],
            consequenceInvolvesEquipmentEntity=[process unit.onto object],
            isConsequenceOfDeviation=deviation,
            isSubsequentConsequence=[scenario[prep.DictName.consequence_2nd]],
            consequenceRequiresBoundaryCondition=process unit.boundary condition)

        consequences = [scenario[prep.DictName.consequence],
                        scenario[prep.DictName.consequence 2nd],
                        scenario[prep.DictName.consequence 3rd]]

        scenario[prep.DictName.severity] = risk assessment onto.SeverityCategory(
                                            isSeverityOfConsequence=consequences,
                                            isSeverityOfEffect=[scenario[prep.DictName.effect]],
                                            severityInvolvesSubstance=[substance.onto_object],
                                            severityInvolvesEquipment=[process unit.onto object],
                                            severityRequiresBoundaryCondition=process unit.boundary condition)

        scenario[prep.DictName.risk] = risk assessment onto.RiskCategory(
                                        involvesSeverity=[scenario[prep.DictName.severity]],
                                        involvesLikelihood=scenario[prep.DictName.likelihood])

        if isinstance(scenario[prep.DictName.cause], ThingClass):
            cause_argument = [scenario[prep.DictName.cause]]
        elif isinstance(scenario[prep.DictName.cause], list):
            cause argument = []
        else:
            cause argument = []

        safeguard = safeguard_onto.Safeguard(safeguardOfDeviation=deviation,
                                             safeguardPreventsEffect=[scenario[prep.DictName.effect]],
                                             safeguardPreventsCause=cause_argument,
                                             safeguardPreventsUnderlyingCause=[scenario[prep.DictName.underlying_cause]],
                                             safeguardMitigatesConsequence=consequences,
                                             safeguardDependsOnRiskCategory=[scenario[prep.DictName.risk]],
                                             safeguardInvolvesEquipmentEntity=[process unit.onto object],
                                             safeguardInvolvesSubstance=[substance.onto object])

        if devex[prep.DictName.safeguard]:
            if safeguard:
                scenario[prep.DictName.safeguard] = [devex[prep.DictName.safeguard]]
                scenario[prep.DictName.safeguard].append(safeguard)
            else:
                scenario[prep.DictName.safeguard] = devex[prep.DictName.safeguard]
        else:
            scenario[prep.DictName.safeguard] = safeguard

    if scenario list:
        sync_reasoner(debug=0)
```

Appendix Listing Q-6: Implemented algorithm for inferring scenarios within propagation mode I

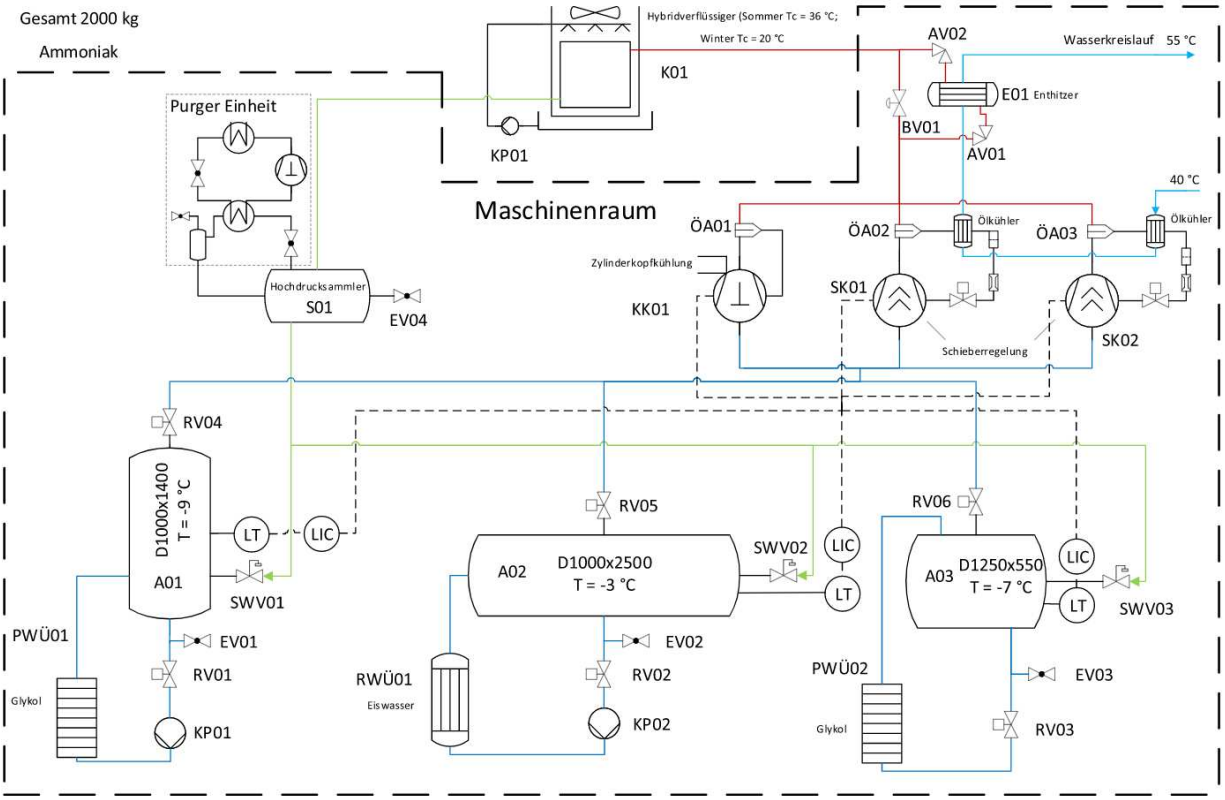
```
# === Check whether cause equals deviation
do not consider = False
no of devs in cause = cause list[0].count(deviation.is a[0].name)
if no_of_devs_in_cause == 2:
    do_not_consider = True

# === Pass results
for scenario in scenario_list:
    consequence list = [scenario[prep.DictName.consequence], scenario[prep.DictName.consequence 2nd], scenario[prep.DictName.consequence 3rd]]

    # Somehow it is not always set above
    if not prep.DictName.safeguard in scenario:
        if devex[prep.DictName.safeguard]:
            scenario[prep.DictName.safeguard] = devex[prep.DictName.safeguard]
        else:
            scenario[prep.DictName.safeguard] = []

    if not do not consider:
        prep.log scenario(0,
                          process unit,
                          substance.name,
                          deviation,
                          scenario[prep.DictName.cause],
                          scenario[prep.DictName.effect],
                          scenario[prep.DictName.underlying_cause],
                          consequence list,
                          scenario[prep.DictName.safeguard],
                          True,  # propagated
                          scenario[prep.DictName.risk])

    # === Infer subsequent deviations
    current_case = {cbr.CaseAttributes.EquipmentEntity: process_unit.onto_object.is_a[0],
                    cbr.CaseAttributes.Event: scenario[prep.DictName.effect].is_a[0],
                    cbr.CaseAttributes.Apparatus: process_unit.apparatus[0].is_a[0],
                    cbr.CaseAttributes.IntendedFunction: process_unit.intended_function[0].is_a[0],
                    cbr.CaseAttributes.SubstancePhase:
                        substance.onto object.hasStateOfAggregation[0].is a[0]}

    match = cbr.match case with cb(current case, cbr.propagation case base)

    # === make sure there is a match and match is not the same as the last one
    # (prevent infinite run due to recursion)
    if match and current_case != previous_case:
        for m in match:
            explanation = process_unit.identifier + ": " + cause[0].is_a[0].name + " -> " + scenario[prep.DictName.effect].is_a[0].name + " -> " + \
                          m.name + " " + "(" + substance.onto object.hasStateOfAggregation[0].is a[0].name + ")"

            subsequent deviation = {prep.DictName.explanation: explanation,
                                    prep.DictName.underlying cause: scenario[prep.DictName.underlying cause],
                                    prep.DictName.safeguard: scenario[prep.DictName.safeguard],
                                    prep.DictName.subsequent_deviation: m(),
                                    prep.DictName.initiating_event: [],   # empty because effects cannot be passed
                                    prep.DictName.likelihood: scenario[prep.DictName.likelihood],
                                    }

            propagation based hazard(subsequent deviation,
                                     process unit,
                                     substance,
                                     last equipment entity,
                                     current_case,
                                     consumed_flag)

            previous_case = current_case
```

Appendix Listing Q-7: Implemented algorithm for inferring scenarios within propagation mode II

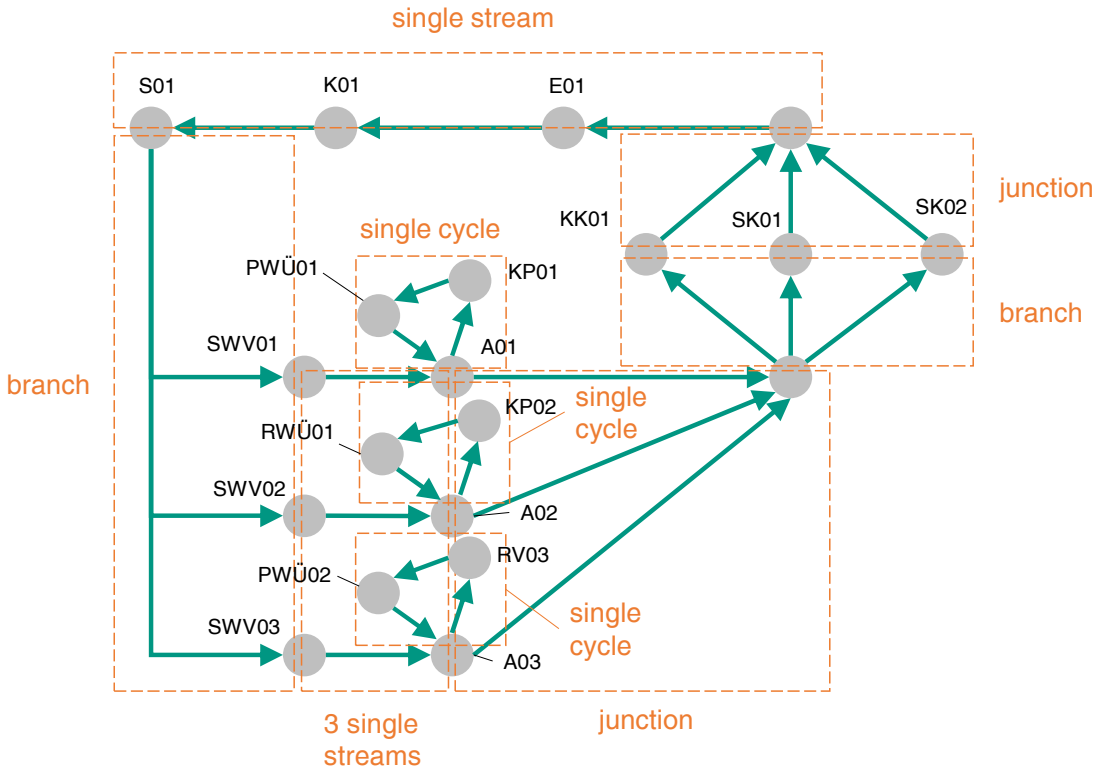
Appendix R Plant section topology related algorithm



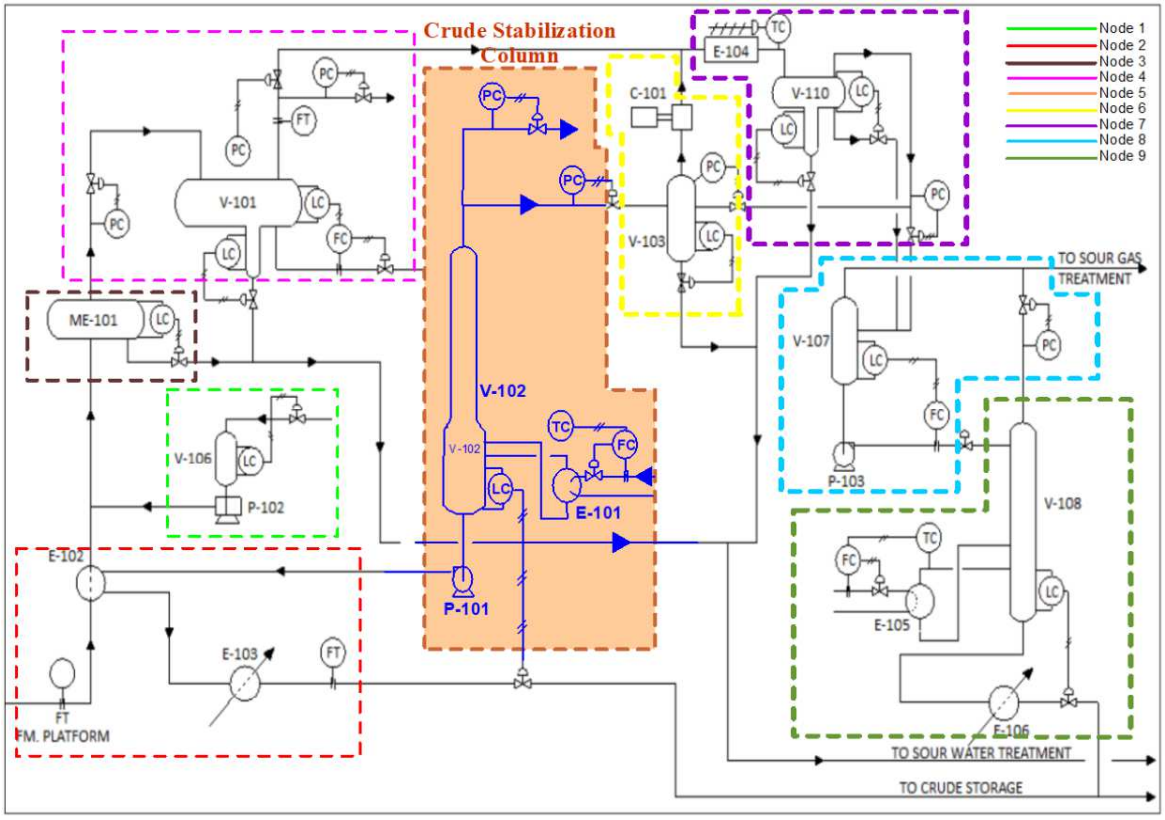
Appendix Figure R-1: Reference P&ID: CSE High Pressure Loop



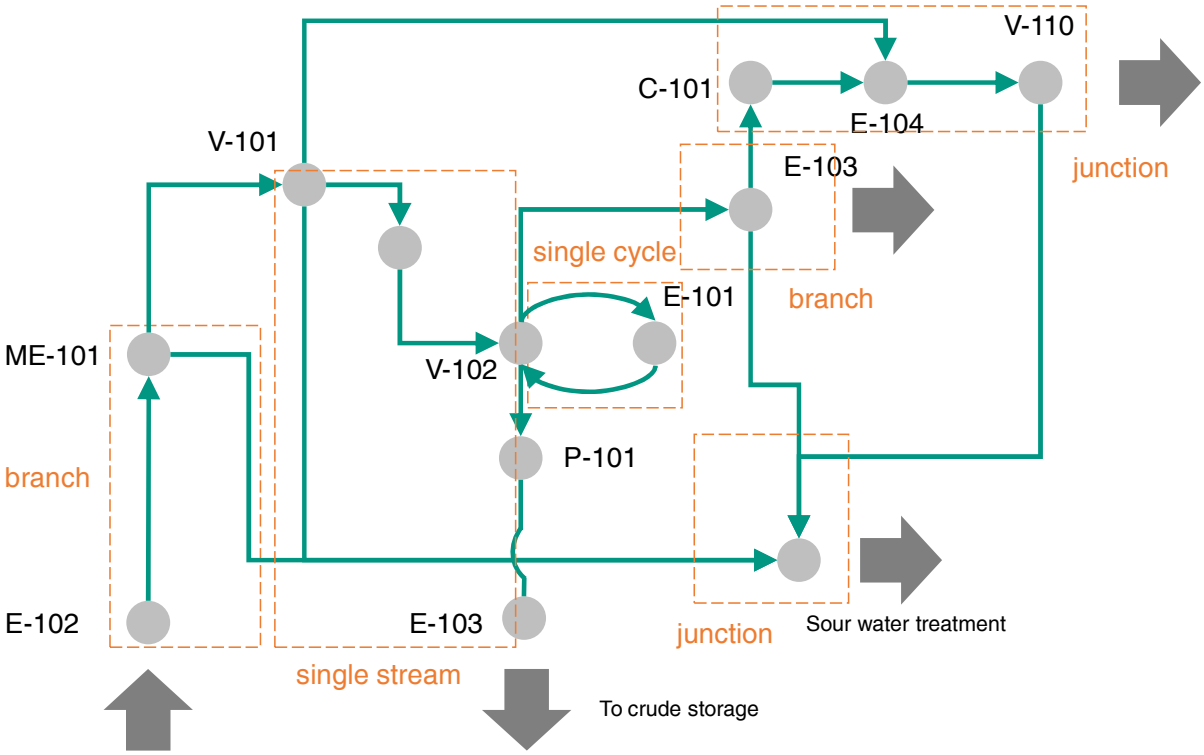
Appendix Figure R-2: P&ID excerpt: Ammonia refrigeration plant



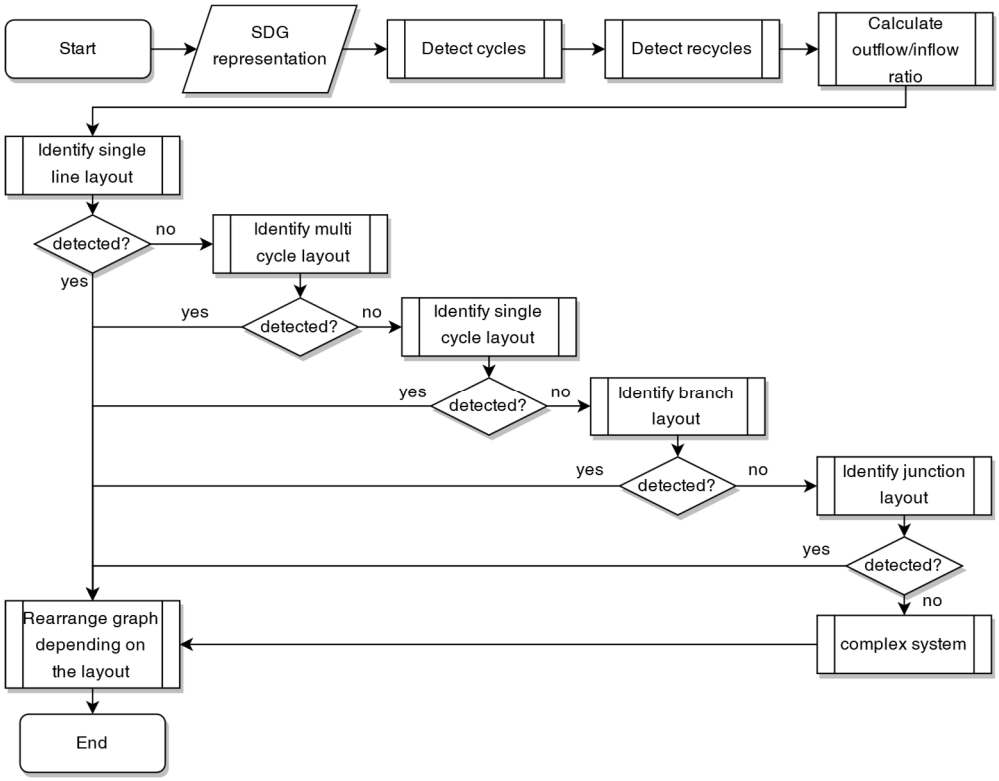
Appendix Figure R-3: Signed directed graph representation of the ammonia refrigeration plant and highlighted basic plant section topologies



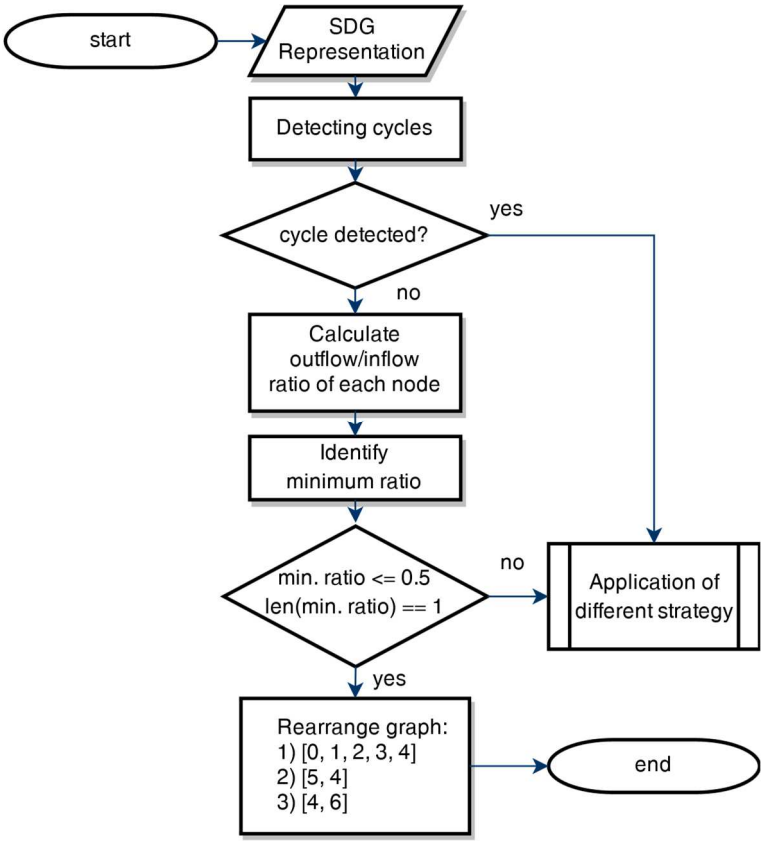
Appendix Figure R-4: P&ID excerpt: Stabilization column (cf. Marhavilas et al., 2020)



Appendix Figure R-5: Signed directed graph representation of the stabilization column plant and highlighted basic plant section topologies



Appendix Figure R-6: Flow chart of the basic plant topology identification algorithm



Appendix Figure R-7: Detection algorithm regarding a junction layout

```
import networkx as nx
from tearing import graph operations as go
from string import ascii lowercase
from enum import Enum

class GraphType(Enum):
    SingleLineSystem = 0
    MultiCycleSystem = 1
    RecycleFlowSystem = 2
    JunctionSystem = 3
    BranchSystem = 4
    SingleCycleSystem = 6
    ComplexSystem = 7


# === Function to identify ratio pattern [0, 1, .., 1, 0]
def identify single line(out in ratios):
    if isinstance(out in ratios, list):
        if len(out in ratios) > 2:
            if out in ratios[0] == 0 and out in ratios[-1] == 0:
                for i in out in ratios[1:-1]:
                    if i == 1:
                        continue
                    else:
                        return False
                return True
            else:
                return False
        else:
            return True


def identify_and_add_predecessors_to_list(graph, current_node, tmp_list):
    end_reached = False
    while not end_reached:
        try:
            tmp predecessor = list(graph.predecessors(current node))[0]
            tmp list.append(tmp predecessor)
            current node = tmp predecessor
        except IndexError:
            end_reached = True
    return


def identify_and_add_successors_to_list(graph, current_node, tmp_list):
    end reached = False
    while not end reached:
        try:
            tmp successor = list(graph.successors(current node))[0]
            tmp_list.append(tmp_successor)
            current_node = tmp_successor
        except IndexError:
            end_reached = True
    return


def my max(e):
    return max(e)

def calc_out_in_flow_ratio(graph):
    node_out_in_ratio = []
    for g in graph.nodes:
        upstream = list(graph.predecessors(g))
        downstream = list(graph.successors(g))

        if len(upstream) == 0:
            node out in ratio.append(0)
        else:
            ratio = len(downstream) / len(upstream)
            node_out_in_ratio.append(ratio)

    return node_out_in_ratio

def identify upstream equipment(graph, reference node):
    return dict(nx.bfs predecessors(graph, source=reference node))

# source https://stackoverflow.com/questions/32997395/iterate-through-a-list-given-a-starting-point
def starting_with(arr, start_index):
    for i in range(start_index, len(arr)):
        yield arr[i]
    for i in range(start_index):
        yield arr[i]

def determine recycles(graph):
    ratios = calc out in flow ratio(graph)

    if all(p == 0 or p == 1 for p in ratios):
        return False
    else:
        cycles = nx.simple_cycles(graph)
        cycles = list(cycles)
        comparison = check equality of list(ratios)
        if comparison:
            return False
        if not comparison and cycles:
            return True
        else:
            return False

def determine cycle intersections(list of cycles):
    list of intersections = []
    for a, b in itertools.combinations(list of cycles, 2):
        # find intersections
        intersection = list(set(list of cycles.get(a)).intersection(list of cycles.get(b)))
        list of intersections.append({"cycles": [a, b], "intersection": intersection})

    return list_of_intersections

# Check whether a list contains the same items
def check equality of list(lst):
    return not lst or lst.count(lst[0]) == len(lst)
```

Appendix Listing R-1: Python helper functions for identifying plant section layouts

```
def determine propagation strategy(graph):
    new graph = []
    type of graph = None
    intersection_node = None
    # === Cycle detection
    cycles = nx.simple_cycles(graph)
    cycles = list(cycles)
    # === Determine recycle streams
    recycle = go.determine recycles(graph)
    # === Calculate outflow - inflow ratio
    ratios = go.calc out in flow ratio(graph)
    # === Identify single line pattern
    single line = identify single line(ratios)
    # === identify roots and potential leaves === see documentation:
    # https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms
    # .simple_paths.all_simple_paths.html
    roots = list(v for v, d in graph.in degree() if d == 0)
    leaves = list(v for v, d in graph.out degree() if d == 0)

    if len(cycles) > 1 and isinstance(cycles[0], list):  # check dimensions
        list of cycles = {}
        # Loop over cycles and assign a letter to each cycle (unambiguous identification)
        for idx, cycle in enumerate(cycles):
            list_of_cycles.update({ascii_lowercase[idx]: cycle})
        # === Determine intersections between cycles
        intersections = go.determine cycle intersections(list of cycles)
        intersection node = intersections[0]["intersection"]

        if len(intersection node) == 1:
            intersection node = intersection node[0]
        # ====================
        for cycle in cycles:
            copy_of_graph = graph.copy()
            for node in graph.nodes:
                if node not in cycle:
                    copy of graph.remove node(node)

            for current, next in zip(cycle, cycle[1:]):
                copy of graph.add edge(current, next)
            new_graph.append(list(copy_of_graph))

        type_of_graph = GraphType.MultiCycleSystem

    elif cycles and not recycle and not single_line:
        new_graph = list(graph.copy())
        type of graph = GraphType.SingleCycleSystem

    elif (recycle or cycles) and len(leaves) == 1:
        # === Tearing procedure in case of recycles
        H = graph.copy()
        # === Calculate outflow - inflow ratio
        ratios = go.calc_out_in_flow_ratio(H)
        # === Evaluate max ratio, position and node
        max_ratio = max(ratios)
        max_ratio_node_pos = ratios.index(max_ratio)

        successors of branch = list(graph.successors(max ratio node pos))
        global list = []

        # === consider first path until diverging node
        node_list = []
        for node in range(max_ratio_node_pos+1):
            node_list.append(node)

        global list.append(node list)
        # === detect nodes that form the recycle
        successor streams = []
        for successor in successors of branch:
            node_list = [max_ratio_node_pos, successor]
            end_reached = False
            current_node = successor
            # === identify all paths after diverging node
            while not end reached:
                try:
                    next successor = list(graph.successors(current node))[0]
                    node list.append(next successor)
                    # === back at diverging node (indicated by maximum ratio)
                    if next successor == max ratio node pos:
                        end_reached = True
                    else:
                        current node = next successor
                except IndexError:
                    end reached = True
            successor streams.append(node list)
        # sort nodes according to numbers (nodes with highest numbers last)
        successor streams.sort(key=my max)
        # Very first list are nodes from first to diverging node
        new_graph = global_list + successor_streams
        type_of_graph = GraphType.RecycleFlowSystem

    elif not cycles and not recycle and single line:
        new graph = list(graph.copy())
        type of graph = GraphType.SingleLineSystem

    else:
        # Check in case all ratios are zero
        if not all(v == 0 for v in ratios):
            num_inflow_nodes = []
            # search for nodes with max inflow streams
            for g in graph.nodes:
                upstream = list(graph.predecessors(g))
                num inflow nodes.append(len(upstream))

            m = max(num inflow nodes)
            max_pos = [i for i, j in enumerate(num_inflow_nodes) if j == m][0]
            upstream_nodes_of_max_pos = list(graph.predecessors(max_pos))
            # === The intention is to identify all streams that lead into the junction
            global_list = []
            for node in upstream_nodes_of_max_pos:
                tmp_list = list(nx.ancestors(graph, list(graph.nodes)[node]))
                tmp list.append(node)  # Append first node before max pos node
                tmp list.append(max pos)  # append max pos node
                global list.append(tmp list)

            # === Consider successors of merged node
            downstream_line = list(nx.descendants(graph, list(graph.nodes)[max_pos]))
            if downstream_line:
                downstream_line = [list(graph.nodes)[max_pos]] + downstream_line
                global_list.append(downstream_line)

            # assemble new structure
            new graph = global list
            type_of_graph = GraphType.JunctionSystem
```

Appendix Listing R-2: Implemented algorithm for identifying process plant configurations I

```
# Idea: in a junction system there is just 1 element where >2 nodes enter and 1 exits
    # this means, just one element has the lowest ratios
    else:
        # === identify minimum ratio
        min_ratio = min(i for i in ratios if i > 0)
        # === get position of minimum ratio
        pos_min_elements = [i for i, x in enumerate(ratios) if x == min_ratio]
        # === check for cycles
        cycles = list(nx.simple cycles(graph))
        # === in case a cycle is detected the plant section is more complex and requires another strategy
        if len(pos min elements) == 1 and \
                min ratio <= 0.5 and len(cycles) == 0 and len(roots) == 1:
            type of graph = GraphType.JunctionSystem

            node_list = list(graph.nodes)
            min_ratio_node_pos = ratios.index(min_ratio)
            min ratio node = node list[min ratio node pos]
            upstream nodes of min ratio = list(graph.predecessors(min ratio node))

            # === The intention is to identify all streams that lead into the junction
            global list = []
            for node in upstream nodes of min ratio:
                tmp list = [node]
                current_node = node
                identify_and_add_predecessors_to_list(graph, current_node, tmp_list)
                # === rearrange global list
                tmp list.insert(0, min ratio node)
                tmp list.reverse()
                global list.append(tmp list)

            # === Search for successors of min ratio node, to find final propagation route
            tmp_list = [min_ratio_node]
            current_node = min_ratio_node
            identify_and_add_successors_to_list(graph, current_node, tmp_list)
            # === Sort results
            global list.append(tmp list)
            new graph = global list

        elif len(leaves) > 1 and len(cycles) == 0 and len(roots) == 1:  # Multiple leaves are an indicator for a branch
            type_of_graph = GraphType.BranchSystem
            all_paths = []
            for root in roots:
                paths = nx.all_simple_paths(graph, root, leaves)
                all_paths.extend(paths)

            new graph = all paths

        else:
            type of graph = GraphType.ComplexSystem
            all_paths = []
            for root in roots:
                for leaf in leaves:
                    paths = nx.all_simple_paths(graph, root, leaf)
                    all_paths.extend(paths)

            all paths2 = []

            for root in roots:
                paths = nx.all simple paths(graph, root, leaves)
                all_paths2.extend(paths)

            new_graph = all_paths

return type_of_graph, new_graph, intersection_node
```

Appendix Listing R-3: Implemented algorithm for identifying process plant configurations II

Appendix S Example unit tests of formal class definitions

```
class TestUnderlyingCauses(unittest.TestCase):

    def test_solar_radiation_1(self):
        cause = causes_onto.AbnormalHeatInput()
        boundary_condition = [boundary_onto.LocatedOutside()]
        super_cause = causes_onto.SuperCause(isSupercauseOfCause=[cause],
                                             supercauseRequiresBoundaryCondition=boundary_condition)

        sync_reasoner()
        super_cause_ = pre_processing.stringify_cleanup_inferred_res(super_cause)
        super_cause_.sort()
        time.sleep(0.01)
        self.assertEqual(super_cause_, ['SolarRadiation'], "Should be ['SolarRadiation']")


    def test_blocked_piping_heat_input_1(self):
        cause = causes_onto.ThermalExpansion()
        unit = equipment_onto.ConnectionPipeEntity()
        boundary_condition = [boundary_onto.ExternalFirePossible()]
        super_cause = causes_onto.SuperCause(isSupercauseOfCause=[cause],
                                             supercauseInvolvesUnit=[unit],
                                             supercauseRequiresBoundaryCondition=boundary_condition)

        sync_reasoner()
        super_cause_ = pre_processing.stringify_cleanup_inferred_res(super_cause)
        super_cause_.sort()
        time.sleep(0.01)
        self.assertEqual(super_cause_, ['BlockedPipingAndHeatInput'],
                         "Should be ['BlockedPipingAndHeatInput']")
```

Appendix Listing S-1: Example Python functions for conducting unit tests regarding formal class definitions of the ontology

Appendix T Ontology metrics

The equations of the ontology metrics developed by (Tartir *et al.*, 2010) and used within this thesis are shown below. The usage of these metrics is also described in (Single *et al.*, 2020d). The relationship richness (RR) is calculated from the number of non-inheritance relationships P, and the number of inheritance relationships (H):

$$RR = \frac{|P|}{|P| + |H|}. \tag{19}$$

The inheritance richness (IR) is calculated from the average number of subclasses per class, and the total number of classes C:

$$IR = \frac{\sum_{C_i \in C} |H^C(C_1, C_i)|}{|C|}. \tag{20}$$

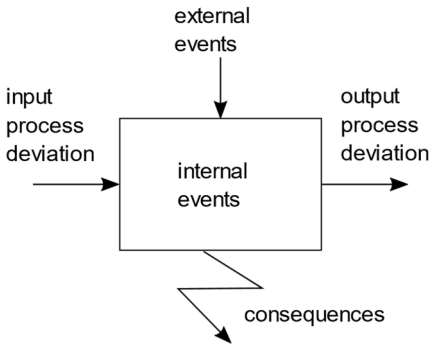
The average population (AP) is calculated from the total number of instances I and the total number of classes C:

$$AP = \frac{|I|}{|C|}. \tag{21}$$

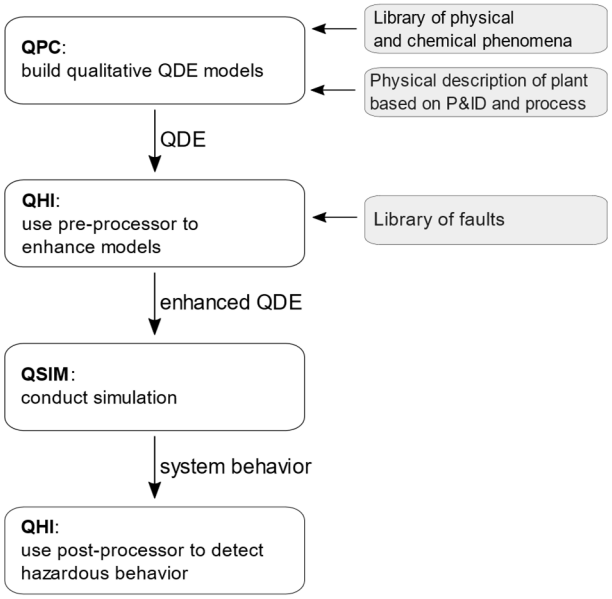
The class richness (CR) is calculated from the number of classes in the base C' and the number of classes in the schema C:

$$CR = \frac{|C'|}{|C|}. \tag{22}$$

Appendix U State of research



Appendix Figure U-1: Model concept of a unit (cf. Heino et al., 1995)



Appendix Figure U-2: Qualitative hazard identifier (QHI) (cf. Single et al., 2019b)

Appendix V Case samples for evaluating case-based reasoning

	No weights		Weights						
No	**SM**	**Correct**	**SM**	**Correct**	**Equipment entity**	**Apparatus**	**Substance phase**	**InitialEvent**	**IntendedFunction**
0	0,6	WAHR	0,85	WAHR	ShellTubeHeatExchangerEntity	PressureVessel	Liquid	AbnormalEvaporation	HeatTransfer
1	0,6	WAHR	0,85	WAHR	ReactorEntity	PressureVessel	Multiphase	LowLevel	Reaction
2	0,4	FALSCH	0,55	FALSCH	AirCooledCondenserEntity	Casing	Multiphase	HighPressure	Condensation
3	0,8	WAHR	0,95	WAHR	StorageTankEntity	AtmosphericStorageTank	Liquid	UnintendedExothermicPolymerization	Storage
4	0,8	FALSCH	0,6	FALSCH	CompressorEntity	CompressorCasing	Gaseous	HighVibration	FluidTransfer
5	0,6	FALSCH	0,85	WAHR	ShellTubeEvaporatorEntity	PressureVessel	Multiphase	NoFlow	Evaporation
6	0,8	WAHR	0,85	WAHR	Pipe	NoApparatus	None	PressureSurge	Transport
7	0,6	WAHR	0,75	WAHR	ValveEntity	Body	None	OtherThanComposition	FluidControl
8	0,4	FALSCH	0,7	FALSCH	PressureReceiverEntity	PressureVessel	Multiphase	HighFlow	Storage
9	0,6	FALSCH	0,85	FALSCH	PlateHeatExchangerEntity	NoApparatus	Multiphase	AbnormalEvaporation	HeatTransfer
10	0,6	FALSCH	0,85	FALSCH	PlateHeatExchangerEntity	NoApparatus	Liquid	HighFlow	HeatTransfer
11	0,8	WAHR	0,9	WAHR	ValveEntity	Body	Liquid	NoFlow	PressureControl
12	0,6	WAHR	0,85	WAHR	WetScrubberEntity	PressureVessel	Multiphase	HighLevel	Purification
13	0,4	FALSCH	0,55	FALSCH	CoolingTowerEntity	OpenVessel	Multiphase	HighFlow	HeatTransfer
14	0,6	WAHR	0,85	WAHR	StabilizerColumnEntity	PressureVessel	Multiphase	HighTemperature	Stabilization
15	0,6	WAHR	0,85	WAHR	TankTruckEntity	PressureVessel	Liquid	NoFlow	Loading

Appendix Listing V-1: 15 sample cases for evaluating the performance of CBR

Appendix W Formal input objects of the case studies

```
def create process plant hexane storage tank():
    hazard classes = [substance onto.FlammableLiquidCategory2, substance onto.SkinCorrosionIrritationCategory2,
                      substance onto.ReproductiveToxicityCategory2, substance onto.SpecificTargetOrganToxicityRepeatedExposureCategory2,
                      substance onto.SpecificTargetOrganToxicitySingleExposureCategory3, substance onto.AspirationHazardCategory1]

    stability_reactivity_information = [substance_onto.FormsExplosiveMixtureWithAir, substance_onto.ReactsViolentlyWithOxidizer]

    substances = [substance.substance("Hexane",
                                      "110-54-3",
                                      substance onto.ProcessMedium,
                                      substance onto.Liquid,
                                      hazard classes,
                                      stability_reactivity_information,
                                      273.15 - 95.0,
                                      273.15 + 65.0,
                                      273.15 - 20.0,
                                      1.1,
                                      8.3),
                  substance.substance("Hexane",
                                      "110-54-3",
                                      substance_onto.ProcessMedium,
                                      substance_onto.Multiphase,
                                      hazard_classes,
                                      stability_reactivity_information,
                                      273.15 - 95.0,
                                      273.15 + 65.0,
                                      273.15 - 20.0,
                                      1.1,
                                      8.3)]

    ambient_information = environment_information.ambient_information(273.15 - 8.0,
                                                                      273.15 + 35.0,
                                                                      [site_information.DangerOfHeavyRainfall(),
                                                                       site_information.DangerOfSevereStorm()],
                                                                      [site_information.VehicleTraffic()])

    boundary conditions = [boundary onto.LocatedOutside, boundary onto.IntroductionOfWater, boundary onto.IntroductionOfImpurities,
                           boundary onto.FoundationCanBeAffected, boundary onto.PersonnelPresentInTheNearField,
                           boundary onto.ExternalFirePossible]

    set_further_boundary_conditions(boundary_conditions, upper_onto.lowest_ambient_temperature)
    boundary_conditions = instantiate_boundary_conditions(boundary_conditions)

    tank_truck = equipment_entities.tank_truck("TT100",
                                               boundary conditions,
                                               equipment onto.Operator(),
                                               False,
                                               [(prep.DictName.intended function, process onto.Unloading),
                                                (None, None),
                                                (None, None),
                                                (None, None)],
                                               1.5,  # bar(g)
                                               273.15 + 80.,  # K
                                               34.4)  # m³
    transfer pump = equipment entities.centrifugal pump 2("P200",
                                                          boundary conditions,
                                                          equipment onto.Operator(),
                                                          False,
                                                          [(prep.DictName.intended function, process onto.DeliverConstantVolumeFlow),
                                                           (None, None),
                                                           (None, None),
                                                           (None, None)],
                                                          1.5,  # bar(g)
                                                          273.15 + 80.,  # K
                                                          90.)  # m³/h
    pipe = equipment entities.connection pipe("PIP1",
                                              boundary conditions,
                                              equipment onto.NotControlled(),
                                              False,
                                              [(prep.DictName.intended_function, process_onto.ModeIndependent),
                                               (None, None),
                                               (None, None),
                                               (None, None)],
                                              15e5,
                                              500)
    tank = equipment entities.storage tank 2("T400",
                                             boundary_conditions,
                                             equipment_onto.NotControlled(),
                                             False,
                                             [(prep.DictName.intended_function, process_onto.Filling),
                                              (prep.DictName.intended_function, process_onto.Storing),
                                              (prep.DictName.intended function, process onto.Emptying),
                                              (prep.DictName.intended function, process onto.ModeIndependent)
                                              ],
                                             0.2,  # bar(g)
                                             360.,  # K
                                             55.)  # m³
    graph = nx.DiGraph()
    graph.add_node(0,
                   data=tank_truck,
                   ports=dict(
                       p1=equipment entities.MyPort('p1', equipment onto.ApiAdaptorValve(), equipment onto.Outlet())),
                   substances=[substances[0]],
                   environment=ambient information)
    graph.add node(1,
                   data=transfer_pump,
                   ports=dict(p1=equipment_entities.MyPort('p1', equipment_onto.InletValve(), equipment_onto.Inlet()),
                              p2=equipment_entities.MyPort('p2', equipment_onto.OutletValve(),
                                                           equipment_onto.Outlet())),
                   substances=[substances[0]],
                   environment=ambient information)
    graph.add node(2,
                   data=pipe,
                   ports=dict(p1=equipment entities.MyPort('p1', equipment onto.InletValve(), equipment onto.Inlet()),
                              p2=equipment_entities.MyPort('p2', equipment_onto.OutletValve(),
                                                           equipment_onto.Outlet())),
                   substances=[substances[0]],
                   environment=ambient_information)
    graph.add node(3,
                   data=tank,
                   ports=dict(p1=equipment entities.MyPort('p1', equipment onto.InletValve(), equipment onto.Inlet()),
                              p2=equipment entities.MyPort('p2', equipment onto.OutletValve(), equipment onto.Outlet()),
                              p3=equipment entities.MyPort('p3', equipment onto.BottomDrainValve(),
                                                           equipment onto.Outlet())),
                   substances=[substances[1]],
                   environment=ambient_information)
    add edge port(graph, 0, 'p1', 1, 'p1')
    add edge port(graph, 1, 'p2', 2, 'p1')
    add edge port(graph, 2, 'p2', 3, 'p1')
    return graph
```

Appendix Listing W-1: Formal input data object of a hexane storage tank

```
def create olefin feed section():
    # Mixture of Hexane/Hexene and water
    hazard classes = [substance onto.FlammableLiquidCategory2,
                      substance_onto.SkinCorrosionIrritationCategory2,
                      substance_onto.ReproductiveToxicityCategory2,
                      substance_onto.SpecificTargetOrganToxicityRepeatedExposureCategory2,
                      substance_onto.SpecificTargetOrganToxicitySingleExposureCategory3,
                      substance_onto.AspirationHazardCategory1]

    stability reactivity information = [substance onto.PolymerizesExothermicallyWithoutInhibitor,
                                        substance onto.PolymerizesExothermicallyWhenExposedToHeat,
                                        substance onto.FormsExplosiveMixtureWithAir]

    substances = [substance.substance("Olefin&Water",
                                      "",
                                      substance_onto.ProcessMedium,
                                      substance onto.Liquid,
                                      hazard classes,
                                      stability reactivity information,
                                      273.15 - 117.5,
                                      273.15 + 63.5,
                                      273.15 - 23.0,
                                      1.15,
                                      7.6),
                  substance.substance("Nitrogen",
                                      "",
                                      substance onto.InertGas,
                                      substance onto.Gaseous,
                                      [],
                                      [],
                                      273.15 - 210.0,
                                      273.15 - 196.0,
                                      None,
                                      None,
                                      None),
                  substance.substance("Olefin&Water&Nitrogen",
                                      "",
                                      substance onto.ProcessMedium,
                                      substance onto.Multiphase,
                                      hazard_classes,
                                      stability_reactivity_information,
                                      273.15 - 117.5,
                                      273.15 + 63.5,
                                      273.15 - 23.0,
                                      1.15,
                                      7.6)]

    graph = nx.DiGraph()

    ambient_information = environment_information.ambient_information(273.15 - 8.0,
                                                                      273.15 + 35.0,
                                                                      [site_information.DangerOfHeavyRainfall(),
                                                                       site_information.DangerOfSevereStorm()],
                                                                      [site_information.VehicleTraffic()])

    boundary conditions = [boundary onto.LocatedOutside,
                           boundary onto.IntroductionOfWater,
                           boundary onto.IntroductionOfImpurities,
                           boundary onto.PersonnelPresentInTheNearField,
                           boundary_onto.ExternalFirePossible
                           ]

    # In case water is not part of the substances it can be introducted by rain, humidity etc.
    for subst in substances:
        if not "Water" in subst.name and subst.intended state of aggregation[0].is a[0].name != "Gaseous":
            boundary conditions.append(boundary onto.IntroductionOfWater)

    boundary conditions.append(boundary onto.UpstreamProcessInvolved)
    boundary_conditions.append(boundary_onto.IntroductionOfAir)

    set_further_boundary_conditions(boundary_conditions, upper_onto.lowest_ambient_temperature)
    boundary_conditions = instantiate_boundary_conditions(boundary_conditions)

    # === Equipment entity definition
    transfer pump = equipment entities.centrifugal pump 2("P200",
                                                          boundary conditions,
                                                          equipment onto.Operator(),
                                                          False,
                                                          [(prep.DictName.intended_function, process_onto.DeliverConstantVolumeFlow),
                                                           (prep.DictName.intended_function, process_onto.ModeIndependent)],
                                                          10.0,  # barg
                                                          400.,  # K
                                                          25.)  # m³/h

    pipe = equipment entities.connection pipe("PIP1",
                                              boundary conditions,
                                              equipment onto.NotControlled(),
                                              False,
                                              [(prep.DictName.intended_function, process_onto.ModeIndependent),
                                               (None, None)],
                                              10.0,
                                              500.0)

    settling tank = equipment entities.settling tank("ST01",
                                                     boundary conditions,
                                                     equipment_onto.NotControlled(),
                                                     False,
                                                     [(prep.DictName.intended_function, process_onto.Separating),
                                                      (prep.DictName.intended_function, process_onto.ModeIndependent)],
                                                     1.0,  # barg
                                                     400.,  # K
                                                     50.)  # m³

    inertization = equipment entities.inertgas blanketing("IB01",
                                                          boundary_conditions,
                                                          equipment_onto.NotControlled(),
                                                          False,
                                                          [(prep.DictName.intended_function, process_onto.Inerting),
                                                           (prep.DictName.intended_function, process_onto.ModeIndependent)],
                                                          2.0,  # barg
                                                          400.,  # K
                                                          )

    control valve = equipment entities.pneumatically flow control valve 1("CV01",
                                                                           boundary_conditions,
                                                                           equipment_onto.NotControlled(),
                                                                           False,
                                                                           [(prep.DictName.intended_function, process_onto.FlowControl),
                                                                            (prep.DictName.intended_function, process_onto.ModeIndependent)],
                                                                           10.0,  # barg
                                                                           400.,  # K
                                                                           )
```

Appendix Listing W-2: Formal input data object of the olefin feed section Part I

```
graph.add_node(0,
               data=equipment_entities.source("Source", boundary_conditions, equipment_onto.NotControlled(), False,
                                              [(prep.DictName.intended function, process onto.ModeIndependent),
                                               (None, None)], None, None),
               ports=dict(p1=equipment_entities.MyPort('p1', None, equipment_onto.Outlet())),
               substances=[substances[0]],
               environment=ambient_information)
graph.add_node(1,
               data=transfer pump,
               ports=dict(p1=equipment_entities.MyPort('p1', equipment_onto.InletValve(), equipment_onto.Inlet()),
                          p2=equipment_entities.MyPort('p2', equipment_onto.OutletValve(), equipment_onto.Outlet()),
                          p3=equipment_entities.MyPort('p3', equipment_onto.BottomDrainValve(),
                                                       equipment_onto.Outlet())),
               substances=[substances[0]],
               environment=ambient_information)
graph.add_node(2,
               data=pipe,
               ports=dict(p1=equipment_entities.MyPort('p1', None, equipment_onto.Inlet()),
                          p2=equipment_entities.MyPort('p2', None, equipment_onto.Outlet())),
               substances=[substances[0]],
               environment=ambient_information)
graph.add_node(3,
               data=control_valve,
               ports=dict(p1=equipment_entities.MyPort('p1', None, equipment_onto.Inlet()),
                          p2=equipment_entities.MyPort('p2', None, equipment_onto.Outlet())),
               substances=[substances[0]],
               environment=ambient_information)
graph.add_node(4,
               data=settling_tank,
               ports=dict(p1=equipment_entities.MyPort('p1', equipment_onto.InletValve(), equipment_onto.Inlet()),
                          p2=equipment_entities.MyPort('p2', equipment_onto.OutletValve(), equipment_onto.Outlet()),
                          p3=equipment_entities.MyPort('p3', equipment_onto.BottomDrainValve(),
                                                       equipment_onto.Outlet()),
                          p4=equipment_entities.MyPort('p4', None, equipment_onto.Inlet())),
               substances=[substances[2]],
               environment=ambient_information)
graph.add_node(5,
               data=inertization,
               ports=dict(p1=equipment_entities.MyPort('p1', None, equipment_onto.Inlet()),
                          p2=equipment_entities.MyPort('p2', None, equipment_onto.Outlet())),
               substances=[substances[1]],
               environment=ambient_information)
# graph.add_node(6,
#                data=equipment_entities.Sink("Sink", boundary_conditions, equipment_onto.NotControlled(), False,
#                                             [equipment_onto.ContinuesProcess, equipment_onto.ModeIndependent], None, None),
#                ports=dict(
#                    p1=equipment_entities.MyPort('p1', None, equipment_onto.Inlet())),
#                substances=[substances[0]],
#                environment=ambient_information)

add_edge_port(graph, 0, 'p1', 1, 'p1')
add_edge_port(graph, 1, 'p2', 2, 'p1')
add_edge_port(graph, 2, 'p2', 3, 'p1')
add_edge_port(graph, 3, 'p2', 4, 'p1')
add_edge_port(graph, 5, 'p2', 4, 'p4')
# add_edge_port(graph, 4, 'p2', 6, 'p1')
# print(graph.edges(data=True))

return graph
```

Appendix Listing W-3: Formal input data object of the olefin feed section Part II

Johannes I. Single

Personal

Nationality	deutsch
Place of birth	Villingen-Schwenningen

Work experience and internships

01/17 – today **Research associate**, *CSE Institut Center of Safety Excellence gGmbH*, Pfinztal.
Research topic: Automation of the Hazard and Operability Method

11/15 – 12/16 **Research engineer**, *CSE-Engineering Center of Safety Excellence GmbH*, Pfinztal.
Tasks: Modeling and software development of design programs in the field of process and plant safety; industrial calculations for pressure relief of single and multiphase flows by means of safety valves with coupled piping systems

03/14 – 02/15 **Academic staff**, *University of Applied Sciences Karlsruhe*, Institute of Applied Sciences, 50 % Part-time job.
Tasks: Planning, development and construction of a test rig for measuring the specific extraction rate of geothermal borehole heat exchangers

05/13 – 02/14 **Intern**, *GEA Brewery Systems GmbH*, Kitzingen am Main,
Department: Refrigeration and CO_2-Recovery.
Tasks: Advancement of software for calculating the cooling load; model improvement on gas purification processes; development of consumption data models for the calculation of key performance indicators (KPI) in breweries.

02/12 – 08/12 **Intern**, *Norwegian University of Science and Technology und SINTEF Energy Research AS*, Trondheim, Norwegen.
Research topic: Experimental investigations of a CO_2 heat pump

10/08 – 09/09 **Surgical Mechanic**, *Karl Storz GmbH & Co. KG*, Tuttlingen.
Tasks: Mechanical manufacturing of medical instruments

Study

04/17 – 02/22 **Ph.D**, *Technical University of Kaiserslautern*,
Department of Mechanical and Process Engineering.
Title: "Automation of the Hazard and Operability Meth-od Using Ontology-based Scenario Causation Models "

03/14 – 10/15 **Master of Science**, *University of Applied Sciences Karlsruhe*,
Mechanical Engineering (Grade: 1,5).
Focus: process engineering, heat transfer and simulation

03/15 – 10/15 **Master's thesis**, *BASF SE*, Ludwigshafen,
Department: Chemical and Process Engineering, Group: Safety Engineering.
Title: "Sensitivity Analysis on Dynamic Process Models"

10/09 – 10/13 **Bachelor of Engineering**, *University of Applied Sciences Karlsruhe*,
Mechanical Engineering (Grade: 1,7).
Focus: Refrigeration, air conditioning and environmental process engineering

08/13 – 10/13 **Bachelor's thesis**, *GEA Brewery Systems GmbH*, Kitzingen am Main,
Department: Refrigeration and CO_2 recovery.
Title: „Development of a model for the economic use of waste heat potential in a brewery"

School and Training

09/04 – 07/07 **Advanced technical certificate and vocational training**, *Feintechnikschule*, Villingen-Schwenningen.
Technical college (Grade: 1,9) and vocational training precision mechanic (Grade: 1,4)

Basic military service

10/07 – 07/08 **Conscript**, *9./LAR, Roth und LIG 13, Landsberg (Lech)*.

Papers

https://scholar.google.com/citations?user=DNgBrR8AAAAJ