

DISSERTATION

Knowledge Representation in Engineering 4.0

*Thesis approved by the Department of Computer Science of the University of
Kaiserslautern for the award of the Doctoral Degree Doctor of Engineering
(Dr.-Ing.)*

to

Dipl.-Ing. Frank Wawrzik

Reviewers

Univ. Prof. Dr. Christoph Grimm
Design of Cyber Physical Systems (WG-CPS)
TU Kaiserslautern

and

Univ. Prof. Dr. Raúl García-Castro
Ontology Engineering Group (OEG)
Universidad Politécnica de Madrid (UPM), Madrid

Dean

Univ. Prof. Dr. Jens B. Schmitt
Distributed Computer Systems Lab (DISCO)
TU Kaiserslautern

Date of Defense: 6th July, 2022

DE-386

Acknowledgements

I want to thank Prof. Dr. Christoph Grimm for his guidance and assistance. I thank him for the freedom that he allowed for following my own research leads and the excellence he conveys in all that he does. He gave my work 'a heart' and I wanted to thank him for supporting me even in bad times of illness. Thank you.

I want to thank Prof. Dr. Raúl García-Castro for the warm welcoming to his research group in Madrid, which I hold dearly and will not forget.

I thank my parents Ernst W. Wawrzik and Ilse A. Wawrzik for their support, love and shelter which accompanied this work in unseen ways. They made me know that parental love is love that is selfless, unconditional and forgiving.

I thank God. May His Will be done.

I wanted to thank all of my chair colleagues for a friendly and supportive work environment. First, Dr. Javier Morena Molina for being my early supervisor and from whom I learned a lot and Dr.-Ing. Carina Zivkovic (born Radojicic) who believed in me. Furthermore I thank Xiao Pan, Thiyagarajan Purusothaman, Sandor Dalecke, Siva Dharsana Chandrasekaran Saratha, Jack Martin (also for being a canteen accomplice), Placide Mucyo, Khushnood Adil Rafique, Axel Ratzke, Marie Madeleine Uwiringiyimana, Johannes Kölsch and Christopher Heinz.

Special thanks to Ralf Grünwald, our technician for helping me out with everyday technical support, being a friend and regular and frequent canteen accomplice, and Manuela Burkart, our secretary for supporting me in the little things that have to be done surrounding work.

Finally, I thank Dirk Denger from AVL List GmbH, which I dearly remember. Who let me be part of his Ph.D. research team and kindly introduced me to his system engineering group and showed me around the company. Also thanks to Peter Neumann from edacentrum GmbH for working with me on ontologies in the project and giving supportive remarks.

Kurzfassung

Diese Dissertation wurde im Kontext des von BMBF und EU / ECSEL geförderten Projektes GENIAL! und Arrowhead Tools entwickelt. In diesen Projekten untersucht der Lehrstuhl Methoden zur Spezifikation und Kooperation in der Automotive Wertschöpfungskette, von OEM zu Tier1 und Tier2. Ziel der Arbeit ist es die Kommunikation und gemeinsame Planung, speziell in den frühen Entwicklungsphasen zu verbessern. Neben SysML ist die Benutzung von vereinbarten Vokabularen und Ontologien in der Modellierung von Requirements, des Gesamtkontextes, Varianten und vielen anderen Elementen angezielt. Ontologien sind dabei eine Möglichkeit, um das Vermeiden von Missverständnissen und Fehlplanungen zu unterstützen. Dieser Ansatz schlägt eine Webdatenbank vor, wobei Ontologien das Teilen von Wissen und das logische Schlussfolgern von implizitem Wissen und Regeln unterstützen.

Diese Arbeit beschreibt Ontologien für die Domäne des Engineering 4.0, oder spezifischer, für die Domäne, die für das deutsche Projekt GENIAL! benötigt wurde. Dies betrifft Domänen, wie Hardware und Software, Roadmapping, Kontext, Innovation, IoT und andere. Neue Ontologien wurden entworfen, wie beispielsweise die Hardware-Software Allokations-Ontologie und eine domänen-spezifische "eFuse Ontologie". Das Ergebnis war eine modulare Ontologie-Bibliothek mit der GENIAL! Basic Ontology, die es erlaubt, automotiv und mikroelektronische Komponenten, Funktionen, Eigenschaften und deren Abhängigkeiten basierend auf dem ISO26262 Standard zu entwerfen. Des Weiteren ist Kontextwissen, welches Entwurfsentscheidungen beeinflusst, inkludiert. Diese Wissensbasen sind in einem neuartigen Tool integriert, das es ermöglicht, Roadmapwissen und Anforderungen durch die Automobil- Wertschöpfungskette hinweg auszutauschen. Ontologien zu entwerfen und zu wissen, wie man diese benutzt, war dabei keine triviale Aufgabe und benötigte viel Hintergrund- und Kontextwissen. Ausgewählte Grundlagen hierfür sind Richtlinien, wie man Ontologien entwirft, Ontologiekategorien, sowie das Spektrum an Sprachen und Formen von Wissensrepräsentationen. Des Weiteren sind fortgeschrittene Methoden erläutert, z.B. wie man mit Ontologien Schlussfolgerungen trifft. Am Schluss wird das Overall Framework demonstriert, und die Ontologie mit Reasoning, Datenbank und APPEL/SysMD (AGILA ProPErty and Dependency Description Language / System Markdown) und Constraints der Hardware / Software Wissensbasis gezeigt. Dabei werden exemplarisch Roadmap Constraints mit dem Automodell verbunden und durch den Constraint Solver gelöst und exploriert.

Abstract

This dissertation was developed in the context of the BMBF and EU/ECSEL funded projects GENIAL! and Arrowhead Tools. In these projects the chair examines methods of specifications and cooperations in the automotive value chain from OEM-Tier1-Tier2. Goal of the projects is to improve communication and collaborative planning, especially in early development stages. Besides SysML, the use of agreed vocabularies and ontologies for modeling requirements, overall context, variants, and many other items, is targeted. This thesis proposes a web database, where data from the collaborative requirements elicitation is combined with an ontology-based approach that uses reasoning capabilities.

For this purpose, state-of-the-art ontologies have been investigated and integrated that entail domains like hardware/software, roadmapping, IoT, context, innovation and others. New ontologies have been designed like a HW / SW allocation ontology and a domain-specific "eFuse ontology" as well as some prototypes. The result is a modular ontology suite and the GENIAL! Basic Ontology that allows us to model automotive and microelectronic functions, components, properties and dependencies based on the ISO26262 standard among these elements. Furthermore, context knowledge that influences design decisions such as future trends in legislation, society, environment, etc. is included. These knowledge bases are integrated in a novel tool that allows for collaborative innovation planning and requirements communication along the automotive value chain. To start off the work of the project, an architecture and prototype tool was developed. Designing ontologies and knowing how to use them proved to be a non-trivial task, requiring a lot of context and background knowledge. Some of this background knowledge has been selected for presentation and was utilized either in designing models or for later immersion. Examples are basic foundations like design guidelines for ontologies, ontology categories and a continuum of expressiveness of languages and advanced content like multi-level theory, foundational ontologies and reasoning.

Finally, at the end, we demonstrate the overall framework, and show the ontology with reasoning, database and APPEL/SysMD (AGILA ProPErty and Dependency Description Language / System Markdown) and constraints of the hardware / software knowledge base. There, by example, we explore and solve roadmap constraints that are coupled with a car model through a constraint solver.

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Kaiserslautern, April 6, 2022



Frank Wawrzik

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Goals and Scope of the Work	2
1.2.1	Problems	2
1.2.2	Goals	2
1.3	New Results	3
1.3.1	Contribution	3
1.3.2	Progress beyond State-of-the-Art	4
1.4	Clarification of Terms	6
1.4.1	What is Engineering 4.0?	6
1.4.2	What are OWL Ontologies?	7
1.4.3	What is SysML?	8
1.5	Vision	9
1.5.1	Semantic Web Stack	11
1.5.2	Web of Thought	12
1.6	Overview	13
2	State of the Art	14
2.1	Ontology Models State-of-the-Art	14
2.1.1	Relevant Ontologies	14
2.1.2	Ontologies in Simulation and System Design	16
2.1.3	Knowledge Representation in Requirements Engineering	17
2.1.4	Context Modeling	19
2.1.5	IoT Ontologies	21
2.2	GENIAL! related Work	23
2.2.1	SysML in Systems Engineering and SystemC	23
2.2.2	SysML vs. OWL	24
2.2.3	Constrained-based Configuration in Expert Systems	28
2.2.4	Self-Aware Cyber-Physical Systems	30
3	Foundations for Knowledge Representation	34
3.1	Basic Foundations for Knowledge Representation	34
3.1.1	From Thesauri to Common Logic	34
3.1.2	From Reality to Ontology	36

3.1.3	Categories of Ontologies and their Meanings	37
3.1.4	Design Guidelines and Methodologies for Ontologies	39
3.2	Advanced Foundations for Knowledge Representation	45
3.2.1	Patterns and Anti-Patterns	45
3.2.2	Multi-Level Theory	47
3.2.3	Reasoning with Ontologies	49
3.2.4	Getting an Overview of Foundational Ontologies	50
3.2.5	BFO Structure and Explanations	51
3.2.6	Part-of Relationship considered more closely	53
3.2.7	Other Developments	54
4	Tools and Methods for Knowledge Management with AgilA	56
4.1	AgilA Introduction	56
4.2	Preparatory Work	57
4.2.1	Initial Architecture	58
4.2.2	Ontology	59
4.2.3	Constraint Net	60
4.2.4	Expert System Implementation	62
4.3	Distributed GUI	65
4.3.1	Refined Architecture	65
4.3.2	Modeling Constraints as Rules	65
5	Knowledge Representation and Models in GENIAL!	67
5.1	Roadmapping Introduction	67
5.1.1	Role of Ontologies in GENIAL!	68
5.1.2	Key Factors for Shaping Microelectronic and Automotive Development	69
5.2	Ontology GENIAL! Modular Structure	70
5.2.1	Basic Formal Ontology (BFO)	71
5.2.2	GENIAL! Basic Ontology (GBO)	72
5.2.3	EFuse Context Model with Solution Spaces	76
5.2.4	Modeling Hardware / Software Allocation for Processors (Prototype)	81
5.3	Formalizing Roadmap Knowledge	81
5.3.1	Starting with some Initial Models	81
5.3.2	Further Models	84
5.4	Hardware Domain	88
5.4.1	Evaluation	88
5.4.2	Application	92
5.4.3	Further Development of APPEL	94
6	Conclusion and Outlook	96
6.1	Outlook	97
A	Appendix	99

A.1	More Complete Reference of Ontologies	99
A.1.1	Abstract and Overview	99
A.1.2	Basic Formal Ontology	100
A.1.3	GENIAL! Basic Ontology	100
A.1.4	Hardware / Software Domain Excerpts	108
A.1.5	Complete APPEL Model for Constraint Exploration	111
A.1.6	AGILA ArangoDB Knowledge Base	117
A.2	Ontology Notation and Legend	122
A.2.1	Basic Elements	122
A.2.2	Classes	123
A.2.3	Object Properties	124
A.2.4	Datatype Properties	126
A.3	The SICYPHOS Framework and a SysML Design Methodology	127
	List of Figures	136
	List of Tables	139
	Abbreviations	140
	Literature	141

Chapter 1

Introduction

1.1 Motivation

To a computer, the Web is a flat, boring world, devoid of meaning. This is a pity, as in fact documents on the Web describe real objects and imaginary concepts, and give particular relationships between them. For example, a document might describe a person. The title document to a house describes a house and also the ownership relation with a person. Adding semantics to the Web involves two things: allowing documents which have information in machine-readable forms, and allowing links to be created with relationship values. Only when we have this extra level of semantics will we be able to use computer power to help us exploit the information to a greater extent than our own reading.

- Tim Berners-Lee "W3 future directions" keynote, 1st World Wide Web Conference Geneva, May 1994

Ontology engineering is an emerging field with increasing number of publications. While not so well understood in the past in general, this field is now considered 'ready' or 'ripe' to find many applications in the industry and beyond. As the author was working in the fields of systems engineering, simulation of cyber-physical systems and artificial intelligence, he became intrigued with what might be achieved using ontologies.

Recent developments in projects showed increasing collaborations across various domains from devices, tools, tool- and value chains and humans and a need to 'unify' and make them inter-operate. We are entering a global information space that needs to be understandable and actionable by artificial agents, tools and devices. Ontologies are the prime way to do that and to open up data from their respective silos. Giving intelligence and meaning to new ways of applications.

1.2 Goals and Scope of the Work

1.2.1 Problems

Semantic Web applications are still not very wide spread and often confined to the IoT to achieve interoperability between devices. The use of ontologies in supply chains is new. Today, semiconductor supply chain collaboration face the challenges of more dynamic changing conditions and complex interactions. Semantic web application to it were for example investigated by [REM22]. In order for a computerised support for the value chain, tools and different participants need to have a common ground on which they are able to operate.

Current ontologies (like e.g. SSN, SAREF) are not able to handle a sufficient covering of the microelectronics domain in a wider sense and in general and the supply chain in particular (P1). They do cover for example only sensors, but not processors, integrated circuits, MOSFETs and other hardware or software. Or they do not cover functions, which are important to the exchange in value chains. SAREF has devices, functions and capabilities but does not express the parts of systems consistently, nor contains software, nor is able to relate the software to the processor on which it runs.

The adoption/breakthrough of semantic web technologies itself faces several challenges. Many of which are out of scope for this thesis, but some are brought to attention in for example [WL21]. *Two are, that some ontologies are itself not interoperable and the quality is not sufficient (they need to be correct) (both combined in P2).* Which renders them useless in a sense or at least to a certain degree.

Current knowledge-based approaches for configuration (e.g. ENGCON¹) may calculate constraints and contain classes of parts with IsA and hasPart hierarchies, but lack the overall framework of the established semantic web in order to be more useful. Those current approaches may be more correctly classified as linked data. *This concretely means that the modelled "knowledge" may likely be quite useless after all in an integrated context, and needs to be flexible and extendable (P3).* Complicated axioms, classifications, conclusions and rules simply cannot be consistently derived.

1.2.2 Goals

Adressing problem (P1), I decided the ontology needs to be based on the well established standard in electronic safety ISO26262 (G1). This for one, makes sure that the terms are already widely adopted and known to its users. Two, that they had an ontological foundation of its definitions. And third, that the ontology would be able to cover the field of microelectronics and systems engineering as a whole, also regarding the requirements for the supply chain and safety. The ISO 26262 standard has so far not been implemented to make it usable for the computer.

Adressing (P2) requires examination and application of top-level ontologies (G2) and methodologies for best practices of ontology engineering (G3). The quality of the ontology needed to be good and it needed to be extensible for the future and in itself

¹<https://sys.cs.uos.de/woru/pub/diplom/html/node25.html>

interoperable.

A further step is then also to build a domain ontology as well as a respective knowledge base with instances and triples in order to evaluate and formally proof the value of the designed ontology (**G4**). Additionally, it is also necessary to acquire data in order to evaluate constraints and their interaction with the knowledge base (**G5**), which finally demonstrates the new overall approach.

Last but not least, related ontologies and reuse of existing ontologies needs to be investigated in order to not reinvent the wheel (**G6**).

Goal (**G6**) in combination with (**G2**) and (**G3**) helped alleviating or minimizing problem (**P3**) by yielding the necessary understanding.

1.3 New Results

The main outcomes or results of this thesis are:

1. The **GENIAL! Basic Ontology (GBO)**
 - *which fulfills the goals G1-G3 and G6*
2. The **GENIAL! Modular Ontology Suite**
 - *which fulfills the goals G4, G5 and G6*

and which contains

- **Hardware / Software Domain**
- **Hardware / Software Knowledge Base**
- **EFuse, Car Model, Innovation, Roadmap and Context Ontologies, Prototypes and more**

3. **An integrated framework for constraint-based knowledge evaluation**
 - *which fulfills the goals G4 and G5*

and which consists of

- **A Knowledge Base in ArangoDB intertwined with an OWL Ontology and a Digital Twin with Frontend and Backend**

1.3.1 Contribution

This dissertation contributes in four main ways:

1. It presents research and representation of a multitude of ontologies/knowledge across a variety of information systems domains and applications in Chapter 2. These domains comprise for example: hardware/software, simulation, automotive, IoT, context, roadmapping, requirements and innovation. The differences of the domains teach how to represent and model knowledge more effectively.

2. It presents expert knowledge of ontology research developments and trends in Chapter 3. Chapters 2 and 3 shall serve as a reference for people working in this field.
3. It applies ontology to the specific field of automotive planning and requirement communication with the development of a prototype tool in Chapters 4 and 5. This is the main contribution and most extensive; ontologies have been modeled and applied to use cases. The tool uses a knowledge base and calculates constraints across the automotive value chain.
4. It is an effort to better understand ontologies and their applications/potential, which can be found throughout the work.

Although many ontology domains are considered, the scope of the dissertation was not to research all information systems domains or one of the mentioned domains extensively. But those related to the engineering 4.0 domain and related domains. A prototype tool was developed, but the focus was on the representation of semantics that the tool uses and to enable modelling with an ontology suite. A more detailed description of the purpose of the project GENIAL! which supported this work is found in Chapters 4 and 5. For reasons of scope and extent (who wants to read a too lengthy dissertation?), some parts were reduced or left out and the author refers to his publications for further details ² or the repository of the ontologies ³. For example, the digital twin [SGW21], the syntax and motivation of APPEL [Gri+21] or a detailed description of the neural net accelerator hardware use case with its constraints [Gri+21] were just outlined. Also, the reasoning evaluation of the GBO ontology was left out [WL21]. Instead, earlier work and beginnings were also selected [WG18][Waw+15] to create a balance. The development of SysMD (interactive notebook) was kept short because it was a recent development.

1.3.2 Progress beyond State-of-the-Art

This dissertation took a research intensive approach to ontology engineering as a whole. Thus, quite some time was spent on vision, state of the art ontologies and application. Compared with state-of-the-art work, this approach and implementation does yield a consistent description for the field of microelectronics and system engineering in general. Its main result, the GENIAL! Basic Ontology (in short GBO), was carefully built with two features in mind. On the one hand, it is based on an upper ontology, which supports modular extension, a hub and spokes approach and a more clear view on how to conceptualize classes. On the other, the axioms are high in expressivity, in order for a most clear usage of the terms and a support of reasoning classification of domains (in this case a hardware knowledge base).

Compared to a SysML-based framework, my approach

- allows for a global data aggregation of all microelectronic components, with a clear enough usage of the terms meaning and classification, to apply rules and exchange

²<https://cps.cs.uni-kl.de/mitarbeiter/frank-wawrzik-msc/seite>

³<https://github.com/wawrzik/GENIALOntologies>

data with others.

- facilitates the use and cooperation with other data communities and integrates in the backbone of the IoT and web.
- is computer understandable rather than just executable.
- is able to harmonize or match different expressions or interpretation of words of different participants.

In comparison to related work, this thesis advances state-of-the-art in several ways:

- It has more expressiveness (Description Logic SHOIQ(D) Expressivity of GBO and HW / SW Domain) than for example Digital Reference (DR) [Ehm+19]. And even many biomedical ontologies, which means, that a word (or a vocabulary) is not just a word as a class in the ontology, but is distinguished in their meaning from other words in a formal way. It contains definitions, metadata, examples, a taxonomy (similar to [CFM19]), and most other ontology axioms.
- It is created in a minimal fashion and contains few object properties as to enhance reuse and facilitate simplicity, unlike [Ehm+19].
- It applies active reasoning for the correct conceptualization of the domain, in comparison to the Semantic Sensor Network Ontology (SSNO) [Com+12], the VICINITY Core Ontology ⁴, the Ontology for Innovation ⁵, the Function Ontology ⁶ and others (e.g. [FM11]). Commonly, most ontologies are connected domain models and still built in this way.
- It is conceptualized more thoroughly with a high quality of the data in mind. It avoids many mistakes of ontologies, that are not built with an intensive background, like the E-Mobility Innovation Ontology [DCV21], which has SubclassOf and InstanceOf errors or suffers from the confusion of instance vs. class etc. Many also confuse domain/range axioms with existential restrictions.
- Combines the roadmap domain with the systems engineering domain in the form of a knowledge base, which hasn't been done before.
- Is integrated into a framework, which combines the concepts of a digital twin [SGW21], a knowledge base (using ArangoDB database), constraints (the ontology served as reference to build a constraint language [Gri+21]) and an interactive notebook. No such solution exists. In few numbers, we find some approaches that successfully combine ontology, knowledge bases and a server client architecture. [Cue+18] is one such example, and also has a more expressive ontology (DABGEO) and an extensive suite for the energy domain similar to this work. However, my work is able to calculate complex dependencies, is interactive and embedded in natural language text documents.
- The approach in this thesis helped advancing reasoners by its specific classification challenges and its unique arrangement of axioms [WL21].

⁴<http://vicinity.iot.linkeddata.es/vicinity/>

⁵<http://purl.org/innovation/ns#>

⁶<https://fno.io/spec/>

1.4 Clarification of Terms

1.4.1 What is Engineering 4.0?

Engineering 4.0 is an emerging term that is - to this date - not yet well defined, and just very recently came into existence.

It is leaned on the term Industry 4.0, which started to take off in 2014 when it became popular and was seen as the overall new direction that the industry should be going toward. Technologies around Industry 4.0 started to develop already about 10 years ago, but were mainly part of conceptional explorations and early trial developments and mostly restrained to the German Research Institute of Artificial Intelligence with its Smart Factory, situated here in Kaiserslautern, Germany. Industry 4.0 is changing the way how we produce and constitutes a new way of industrial revolution, that incorporates a significantly enhanced and interdisciplinary process and workflow and works throughout the production, maintenance, operation and recycling cycle.

Engineering 4.0 is the process and methodology that comes from the new technologies and tools that are used in and stem from the paradigm of the Internet of Things and Industry 4.0. Though it is an emerging term, what to this day can probably already be said is that it will be based on a common shared international vocabulary in the form of ontologies. Which purpose is to expose the data from different tools, models and data silos and to enrich its meaning, in the way that it does not only become machine readable, but also machine understandable. In that way it becomes possible to utilize data across company boundaries and beyond specific disciplines. Furthermore Engineering 4.0 will stand for a more agile, collaborative approach with increased communication between various parties and a focus on product-service systems.

The term will be based on more classical disciplines in the beginning. Figure 3.15 gives an overview. Systems engineering (i.e. connectedness of engineering processes and sys-

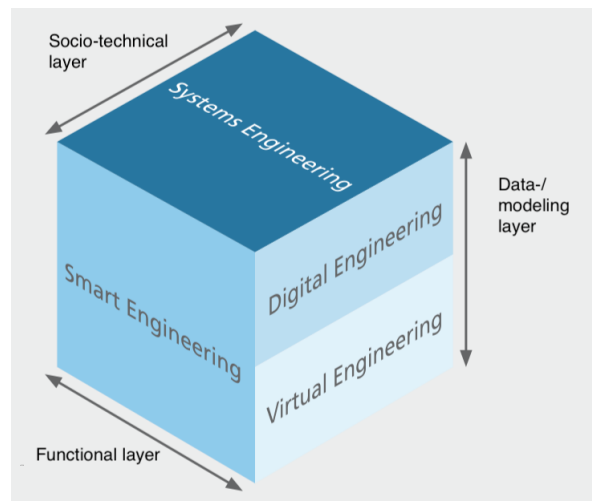


Figure 1.1: Engineering 4.0 Overview ([Mat16], modified)

tem integrator), digital (i.e. digitizing documents, creating online tools), virtual (e.g. simulation of hard-, software and mechanical parts) and smart engineering (e.g. networked embedded systems that are part of processes) are all part of this development. Furthermore here is a summary of aspects that are important to Engineering 4.0, that are also part of the project 'GENIAL' and the tool 'AgilA' worked out in this thesis:

- High level of abstraction needed
- Hiding certain knowledge from competitors (to compete as well as cooperate): Black and white boxing of knowledge in iterations and in dialogue with ontologies as interfaces
- Gathering of relationships and relational knowledge and alternatives that constrain the solution space of particular constructions
- Detailing knowledge (refining it) within a company with certain user rights
- Tracking knowledge needed, which enables to relate decisions made on requirements (and see what options were chosen why and which ones have been discarded)
- Disambiguating interpretation of terms
- Interfacing with tools (e.g. Polarion/Doors)

1.4.2 What are OWL Ontologies?

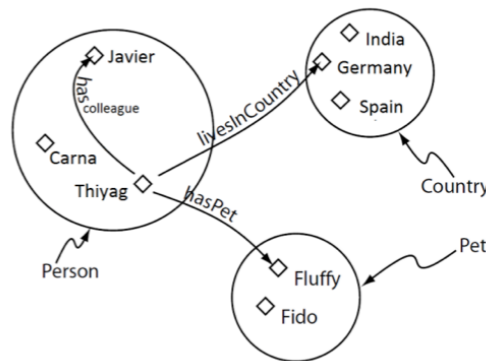


Figure 1.2: Ontology Example ([Hor11], modified)

An ontology is a specification of a conceptualization, a formal, explicit specification of a shared conceptualization [Gru95] or a formal specification of a domain of discourse. In other words an ontology allows to represent and model certain kinds of data by using words (semantics) which indicate their meaning and their relationships. They are captured in the Ontology Web Language (OWL). In contrast to writing code, the semantics of ontologies are usually directed to using natural language expressions in order to achieve a higher reusability of their vocabulary. The modeling constructs of

ontologies are classes, individuals, data properties and object properties, which have some relation to general programming languages (classes, objects/instances, integers/strings, properties).

Figure 1.2 gives an example. Here it can be seen that there are different kinds of persons (instances (individuals) of the class person), which stand in direct relationship (object properties) to other individuals. As can be seen the vocabulary is chosen in a consistent way so that the knowledge would be easily extensible and put in relation to other objects and reused. By modeling data in this way it becomes possible to ask questions about that data. For example now it could be asked which persons live in the country Germany? And with a corresponding SPARQL query the answer would be that the person 'Thiyag' lives there. Otherwise another question could be formulated to 'which persons have colleagues and at the same time own a pet called Fluffy'. In this way it becomes possible to give more meaning to data, and not just leave it as raw data which might only be processed by specific tailored computer programs.

1.4.3 What is SysML?

SysML [Sta15] (the System Modeling Language) arose as a need from more connected engineering processes and an increased collaboration between peers. Furthermore it was supposed to bring together and manage everything for a project engineer or system integrator. It is a graphical modeling language to represent hard-/software, mechanical and business artifacts also covering requirements and was sought to address the three following issues:

- Documentation
- Heterogeneity
- Modeling

It was supposed to digitize data from systems engineering tools and processes, illustrate them in one language or framework and even use them for simulation and verification. SysML has an emphasis on representing the relationships of various views and design methods. Figure 1.3 shows which kind of parties are supposed to be covered and integrated within the language.

SysML takes some of its components from UML (Unified Modeling Language), modifies some of it and provides some of its own constructs. Figure 1.4 gives an overview of the language. It consists of a requirements diagram which allows for text based documentation of informal requirements and their relationships. It has a block definition and internal block diagram which allows to display the structure of components and their connections and parameters (hard-/software, mechanical). The activity diagram

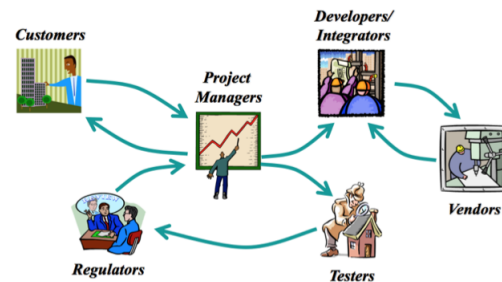


Figure 1.3: SysML Motivation [San09]

specifies a controlled sequence of action with routing flows and decision, join and merge nodes. The sequence diagram represents message based behaviour, the flow of control and interaction of parts. The state machine diagram consists of nodes, states and conditions under which states are invoked. The package diagram makes it possible to sum up diagrams in packages and give a namespace. The use case diagram involves people and allows to allocate who is working on or responsible for which use case.

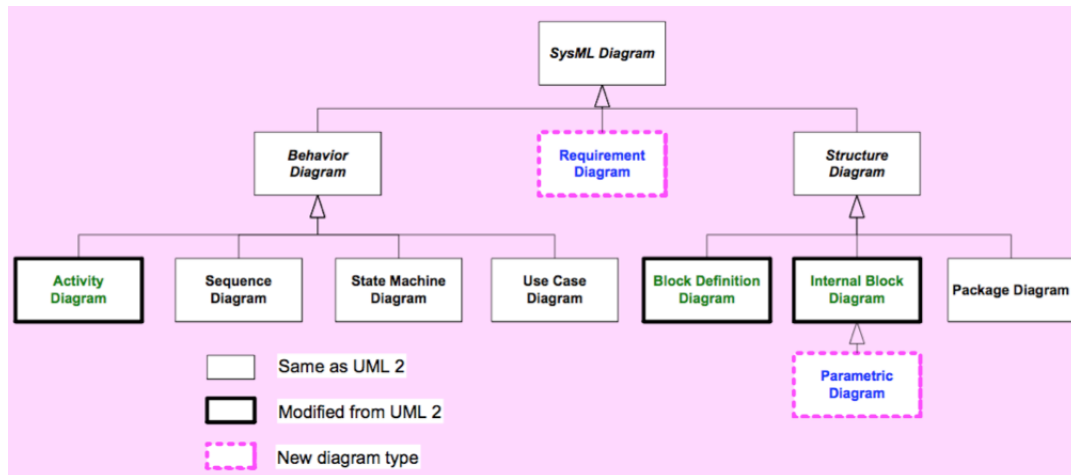


Figure 1.4: SysML Overview [San09]

1.5 Vision

Finally the vision of future developments shall be mentioned as some parts of this thesis work is already touching upon some subjects that are only to be addressed adequately in the future. For example, let's look at the IoT pyramid, which has some similarity with the Maslow's hierarchy of needs. And indeed, as we will see some areas of research blurring and intersecting, we will see many areas converging. Here a few developments shall be put into context and let's look at figure 1.5.

For example we haven't really even closely arrived at the big data level, which is at the bottom of the chart and the basis for further developments in the hierarchy. Before we can put meaning to data, we will need a vast amount of data in the first place. 'There are many sources that predict exponential data growth toward 2020 and beyond. Yet they are all in broad agreement that the size of the digital universe will double every two years at least, a 50-fold growth from 2010 to 2020.'⁷ 'YouTube receives roughly 300,000 individual video uploads each day, amounting to 80k hours of video and 24TB of data (one billion users and one billion hours of video watched daily.'⁸ 'Every 60 seconds on

⁷<https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/>

⁸<https://www.youtube.com/about/press/>

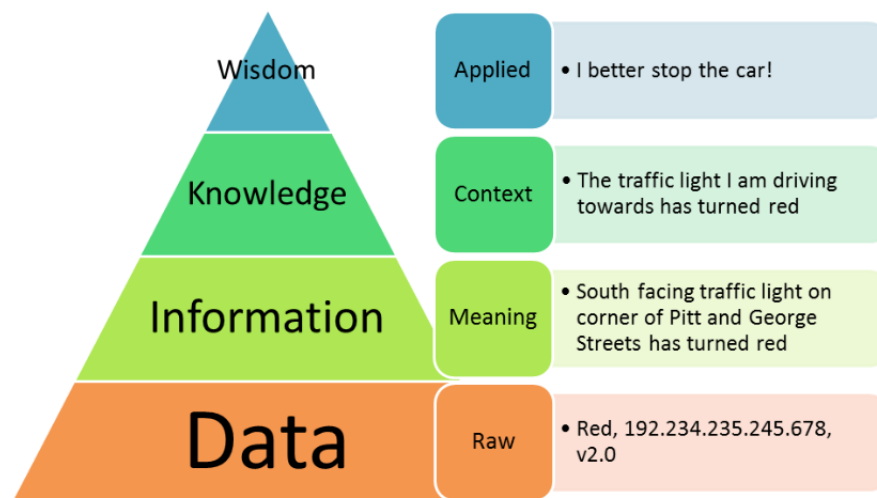


Figure 1.5: IoT Direction [Søn16]

Facebook: 510,000 comments are posted, 293,000 statuses are updated, and 136,000 photos are uploaded.⁹ The IoT with its sensors will be significantly contributing to the data growth. After the level is enriched with meaning we will then be able to talk about and work more with putting things in the correct context (knowledge level), and context knowledge in general. Autonomous agents are also part of this level.

Personal software agents will be enabled through semantic technologies and organize personal calendars, coordinate activities with family, friends and acquaintances and answer simple requests from sources outside the owner. For example a personal agent will be given a task to buy a suitable car. He knows the preferences of the owner and selects a preliminary choice. The agent contacts various other agents of different services and finally selects a few services that are nearby. They communicate prices, product details and delivery conditions and negotiate autonomously. By doing that the agent considers reputation of providers with the help of independent sources. If an offer is found the agent accesses the personal calendar and finds a date for a test drive and plans it. Finally the agent informs the owner through a personal device he has at hand (paraphrased from [Woo02]).

Though self-driving cars are currently being developed and have achieved some success and functionality¹⁰, it will take some more time until they become semantic agents, and can react smartly under certain circumstances. Only if the previous stages are set, we are able to address what it might look like for an agent or system to behave wisely.

⁹<https://zephoria.com/top-15-valuable-facebook-statistics/>

¹⁰<https://www.wired.com/story/when-will-self-driving-cars-ready/>

1.5.1 Semantic Web Stack

The semantic web community seems to be the most advanced when it comes to modeling knowledge and its semantic web stack with its ontologies was thus chosen as the focus of knowledge representation of this thesis. The semantic web stack is illustrated in figure 1.6¹¹.

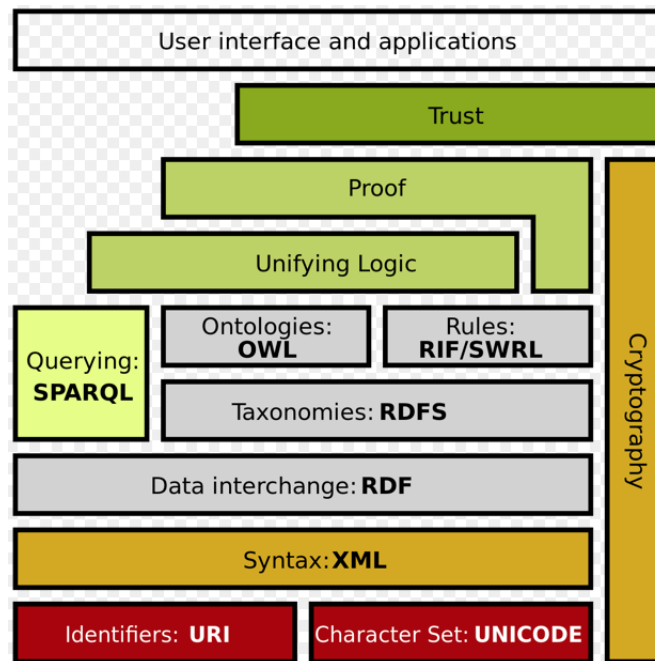


Figure 1.6: Semantic Web Stack Overview [Wik18]

At the bottom of the figure we have unique resource identifiers (URIs) which give each modeling construct an address to refer to. This makes each concept identifiable and referable which supports a global use of these constructs. Of course the stack is based on Unicode which is a set of characters to express everything in words and numbers in different languages. RDF, the resource description framework is a first way to define relationships and links between a subject and object, also known as triples. In comparison to ontologies it uses resources, URIs and literals and can be more classified as linked data than the well defined meaning of ontologies. RDFS, RDF Schema, yields a vocabulary to describe RDF and thus closes the gap towards OWL. SPARQL is the query language for OWL. The user interface is on top of the stack and is typically a Java program. It makes use of the knowledge and visualizes it.

The semantic web stack is put into vision as especially the upper parts of the stack have not yet been fully realized. Even to the unifying logic layer there is little to no

¹¹https://en.wikipedia.org/wiki/Semantic_Web

information to what it actually is or supposed to be. It may only be known that this layer may have some correlations with the goal of a strong A.I. There are some preliminary implementations of the proof and trust layer, which are presented in [Sta07] [Hen08]. In addition the development of these technologies is also a *social process*. For example in earlier development stages of the semantic web there was not so much of a willingness to share one's own data in an open way which is still an issue and comes down to the readiness of companies to cooperate instead of compete. So there are actual barriers of ones own evolution preventing the breakthrough of these technologies. In discussion with peers at conferences, it came out that this today is still quite an important issue to be overcome for a further progression of the semantic web.

1.5.2 Web of Thought

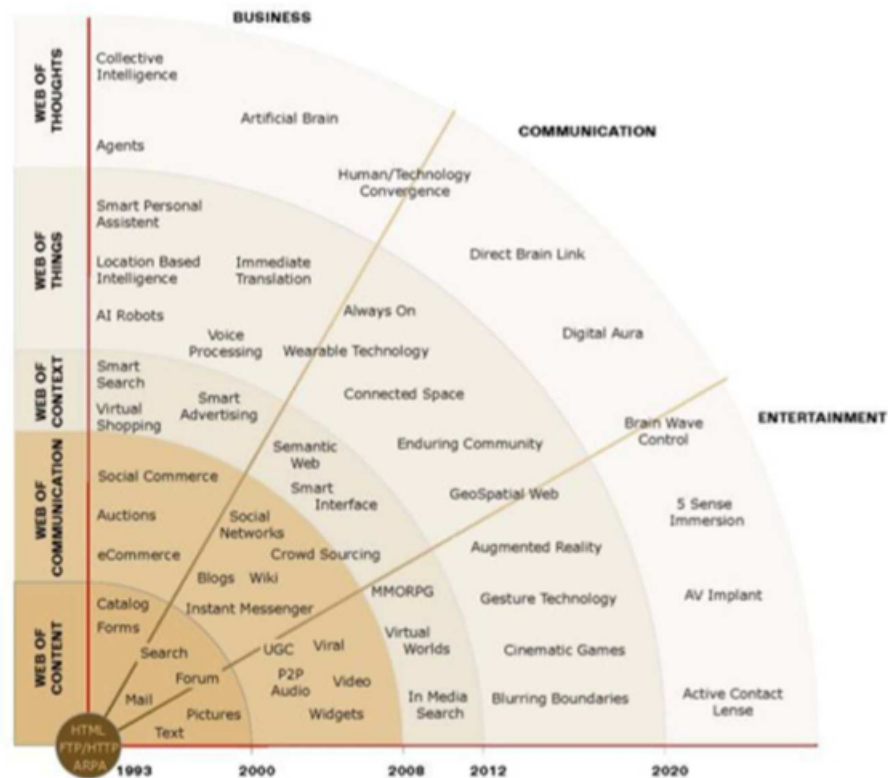


Figure 1.7: Web of Thought Vision [TL16]

The web of thought is an advanced vision, which is already worked on intensively. Figure 1.7 shows some of its developments. As can be seen the figure contains many converging, exponential technologies from disciplines like biology, artificial intelligence, virtual reality and esoteric concepts (like digital aura). One prominent example may be Elon Musk's

Neuralink, which works on a direct brain link. But also the artificial brain, or hive mind are significant developments. There are also examples that disabled people are able to steer simple computer games or devices via brain wave control. All these are very potent technologies and ontologies will play a role in realizing and supporting them.

1.6 Overview

Chapter 2 gives an overview of state of the art of knowledge representation in various areas. It addresses ontologies in the domain of simulation, hard- and software design, context, innovation, IoT, requirements and others. Chapter 2 also addresses SysML in the modeling of hard- and software with an emphasis on co-design. To show the present state-of-the-art a comparison of SysML and OWL is given.

Chapter 3 gives a summary of the authors research about ontologies in a general way. First the basics are introduced to give an understanding of the technology. This comprises different ways to represent knowledge on a continuum of technologies and languages, the various ontology types, and guidelines in designing ontologies. Second, advanced foundations are presented. They encompass reasoning, multi-level theory (as an example to go beyond current language standards), patterns and anti-patterns to make an ontological analysis, an overview of foundational ontologies and recent lesser-known developments. In chapter 4 the work of the author starts. It describes the work about the tool AgilA, a new methodology to design automobiles with just their semantic features and calculation functions by restricting constraints during the construction. An expert system was integrated into the knowledge base.

While in Chapter 4 the tool is described in its first prototype and the emphasis is placed on how the knowledge utilized by AgilA is used, Chapter 5 focuses on the representation of the semantics that the tool uses. A modular ontology suite and GBO with its classes, properties and definitions is introduced and applied to an eFuse and HW / SW use case with additional other prototypes and examples. Finally in this chapter roadmap knowledge has been modeled to examine applications of the tool and possible interactions with the design.

In the appendix, first the resulting ontology is outlined in more detail. Definitions and classes, properties and instances are provided as well as an overview. Next the SICYPHOS framework of the chair 'Design of Cyber-Physical Systems' is introduced which was enhanced with a semantic SysML cross-domain design approach.

Chapter 2

State of the Art

In the following, the chapter of state of the art is intentionally stated broadly, as to indicate the authors alignment and span the space in which the author can move and intensify research if the project requires or as to be able to find new solutions to existing approaches from different disciplines. As stated previously the area of ontologies was given more attention over SysML as they seem more complex and significant as technology advances.

2.1 Ontology Models State-of-the-Art

2.1.1 Relevant Ontologies

From the author's paper about roadmapping that analyzed a variety of ontology models [Waw+]: *"There seemed to be either relatively few, or none, ontologies out there that described roadmap information or general domains of interest like artificial intelligence, e-mobility, demographic development or legal boundary conditions. It would seem rewarding if this kind of knowledge would also be formalized, not only maintained by the GENIAL! roadmap manager, but by the research institutes that are working in those fields and updated in a distributed way so that parts of it could be reused by the Agila tool.*

There were few ontologies that you find in the automotive [FM11] and none in microelectronic or semiconductor domain. That is in stark contrast to the vast ontologies already developed in the internet of things domain. E.g. LOV4IoT¹ lists, at the time of writing, a catalog of 548 ontology-based research projects for IoT and its application domains, like health, environment, transport and other topics. It is comprehensive and well documented, also listing corresponding papers, the OWL files, their status, and additional information.

In the automotive sector, there's the W3C Automotive Ontology Working Group², which is a rather informal group that intends to create shared OWL vocabularies for the automo-

¹<http://lov4iot.appspot.com/?p=ontologies#transport>

²<http://www.automotive-ontology.org>

tive industry and extend *schema.org* for search engines. Most of its work thus far seems to be planned, and its activity seems rather stagnating. There was the *CRYSTAL* project which researched existing automotive ontologies³. It coped with the questions of how an automotive ontology should be represented, what should be the scope and on what sources it should be based. The results in the above mentioned link are scarce. *Auto-schema.org*⁴ lists basic classes and properties of cars. It is affiliated with the W3C Automotive Ontology Working Group. The *autoschema* vocabulary seems to serve as *RDFa* to be embedded in websites, to use it for e.g. rich snippets, rather than as *OWL* vocabularies. Additionally, there is the vehicle signal and attribute ontology [Klo+18], which contains a multitude of different signals that are usually contained in different formats and belong to various architectures. It aims to provide interoperability by being implementation independent. A *SAREF* extension for automotive⁵ is a recent development which treats automobiles as an extension of the *IoT*.

Currently there are several, but few upper/foundational/top-level ontologies that serve different purposes and take a considerable amount of time to understand as they introduce quite some complexity. Research shows that there are benefits in using top-level ontologies [Sch18] and that they outweigh the disadvantage of complexity [Kee11]. Some benefits that seem to justify the usage of an upper level ontology for *GENIAL!* are:

- Interoperability
- Facilitates best practices
- More classes less object properties

As more ontologies become available, we move up in the hierarchy and context becomes the determining factor. Context ontologies are in their early stages and constitute yet very basic concepts. [Wan+04] proposes *CONON* (*CONtext ONtology*), a first context ontology, which models basic concepts like person, location, activity, and computational entity to specify context. It furthermore consists of specific domain ontologies like smart home and smart office that become integrated in that ontology. *CONON* can be seen as a top level ontology (*TLO*), which is one way to model context knowledge because they usually use a very broad terminology. *CONON* uses user- and ontology defined reasoning capabilities to deduce implicit context. *CaCOnt* [Xu+13] is a more recent context ontology which builds on *CONON*. It has a more refined user model, device model, service model, space model, and environment model. It also has similar reasoning capabilities. [CFM19] is a more elaborate and comprehensive work, which takes the concept further. It presents *3LConOnt*, a three-level ontology for context modelling in context aware computing. It is structured in top-level, middle-level and lower-level and reuses a manifold of ontologies, ranging from regulations, environmental conditions to roles, objects, activities and resources. 13 context-aware scenarios are described which could be instantiated with this

³http://www.crystal-artemis.eu/fileadmin/user_upload/Deliverables/CRYSTAL_D_308_010_v1.2.pdf

⁴<https://auto.schema.org>

⁵https://www.etsi.org/deliver/etsi_ts/103400_103499/10341007/01.01.01_60/ts_10341007v010101p.pdf

ontology. It does not reuse BFO⁶ (Basic Formal Ontology, introduced later) and varies from it in the upper level by its conceptualization of separating context information from entity, which is also adapted in this work. Though it has some relevant classes, BFO seemed more suitable.

The source in⁷ describes the ontology for innovation developed by Volkswagen AG. It is valuable for the project in the sense that it one allows to describe needs, problems, development stages, disruption and usage of innovations. It is general enough to describe all kinds of innovations and its content may influence solution spaces. E.g. different innovations may solve the same problem in different ways. The GenID Ontology [BAS17] is a generic modular ontology, which is supposed to solve the issue of interoperability in the innovation domain. One of its modules is also conceptualizing a domain of context. [Rie+09] presents an ontology of ideas which is closely related to the open innovation domain.

[Fah+18] shows an ontology that is able to allocate hardware resources to computationally intensive BioProblems which are implemented by certain computer science algorithms. As related work OntoCape⁸ [MYM07] from university RWTH Aachen needs to be referenced. It is a comprehensive work for computer aided process engineering. Though it describes chemical processes, it also has firm models rooted in system theory that support integration and reasoning, mathematical models, upper level, supporting and meta concepts."

2.1.2 Ontologies in Simulation and System Design

The field of applying ontology modeling in simulation is still in its infancy [Gro+12]. The goal is to facilitate reuse of existing models, compare them, make them queryable and making inferences. [Gro+12] implements an ontology in the simulation domain where the simulation models are instances of the ontology. It consists of a layered architecture with an upper ontology for terms that are common among simulators, a simulators' ontology layer that has an ontology specific to the corresponding simulator and a ontology based simulation models layer. Represented are the simulation models hierarchies and connections between modules. They use SPARQL and SQWRL to query their models and their approach has several advantages as they are using existing domain models to generate their ontology. This approach is exemplified in figure 2.1. As can be seen a transformation engine reads the ontology as well as the simulation models by creating the corresponding instances in the ontology; at the end the updated ontology is written to the .owl file and gives the semantic representation of the simulation models. The integration is realized via OWL API and the simulation environment is built upon MATLAB Simulink. It is notable that SQWRL was more suited to perform the required queries since it is an ontology query language it was able to make inferences and find results attributed to superclasses.

⁶<http://purl.obolibrary.org/obo/bfo/2.0/>

⁷<http://www.lexicater.co.uk/vocabularies/innovation/ns.html>

⁸<https://www.avt.rwth-aachen.de/cms/AVT/Forschung/Software/~ipts/OntoCape/lidx/1/>

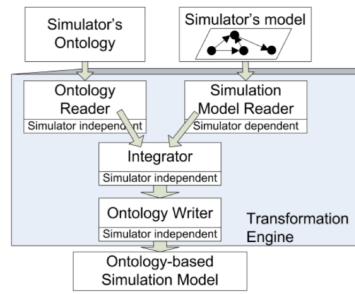


Figure 2.1: Ontology Integration with Simulation Models [Gro+12]

In the following there is a strongly comprised summary of approaches in simulation of hardware design and synthesis from the paper "A Concept for Design of Embedded Systems at Semantic Level" by the author for reference [WMG14]: "In particular in the synthesis of analog circuits, knowledge-based systems are used for configuration of analog circuits (e.g. OASYS [HRC87], KANDIS [OGW95]). In these tools, the knowledge-based design approach targets structural refinement of analog circuits, but not modeling of HW/SW interfaces. IP-based design aims at supporting re-use considering parameterization changes [JWS01], IP enhancement and database management [SSS05] and simple ontologies to structure libraries [ZT13]. In particular IP-XACT [Her+12] allows describing register maps. While IP based re-use increases productivity, it has not been used for abstracting communication and generating new models. Knowledge-based approaches and ontologies for modeling/simulation are a quite new topic [Gro+12]. Eriksson [Eri+18] proposes an approach for simulation of infectious diseases with ontologies, separating modeling from code and execution. Communication abstraction and synthesis have not yet tackled the idea presented in this paper. [Pen+06] [Car+12] infer various layer block components like buses and protocols; generate C code and abstract communication to the service level. However, [Pen+06] [Car+12] generate general communication functionality, but not register maps of peripherals and the referring software. Compared with previous work, this work allows modeling in an interactive way comparable with Matlab/Simulink, while being able to immediately start HW and SW development.'

2.1.3 Knowledge Representation in Requirements Engineering

Requirements are usually captured informally in tools like Doors or Polarion. Or to give some more structure to it they are captured in SysML with explicitly stating some dependency relations, where they can also be linked to specific building blocks in the implementation. It is relevant to formalize requirements in order to make them usable for machines that are then able to configure a product or system on the fly depending on the changing requirement. Ontologies are being introduced more and more to capture requirements and their dependencies. There are several issues that have to be addressed in requirements engineering that are simultaneously a strong point for using ontologies. An older paper states the issues that are still relevant today [LFB96]:

"There is a need for a representation of requirements for engineering design that:

- Provides unambiguous and precise terminology such that each engineer can jointly understand and use in describing requirements.
- Allows traceability of the requirements, with dependencies and relationships among the requirements captured and stored.
- Support the detection of redundant or conflicting requirements.
- Is generic, reusable and easy to extend
- Integrates requirements with parts, features, parameters and constraints.
- Facilitates document creation conforming to customer/company/government rules and regulations.
- Facilitates the change management process"

Many papers address the basics of evaluating the usefulness of ontologies in requirements engineering, their benefits and opportunities [Cas+10] [Sie+11]. [LFB96] introduces a product, feature and parameter ontology and basic axioms for reasoning (e.g. that a part cannot be a component of itself, or that a part cannot have a component which has as a component the first part, or that a subfeature of a feature of a part is also a feature of that part). Furthermore, the paper describes formalizations of derived, explicit and sourced requirements with some classifications.

[Cas+10] gives a general review of ontologies in requirements engineering and shows some benefits. It lists ontologies for describing requirements specification documents, ontologies for formally representing requirements and application domains.

[Sie+11] examines with a framework if goal oriented requirement models are consistent and complete when they are communicated from stakeholders. It introduces rules for completeness and consistency checking.

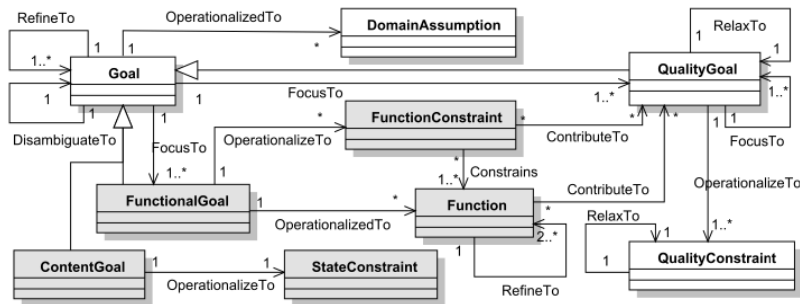


Figure 2.2: The Extended Requirements Ontology [Li+15]

[Li+15] goes into much more detail of ontological distinctions of requirements and enables translating informal requirements to formal ones with the help of an ontology and an intermediary requirements specification language. This work showed that it was possible to even model, analyze and consistently represent highly informal knowledge. It

also addresses non-functional requirements, seldom addressed in other works. The main contribution is probably the ontological classification of requirements and the resulting requirements ontology shown in figure 2.2.

The classifications of functional, usability, operational, security, performance, availability, look and feel, maintainability, scalability, portability, fault intolerance could all be represented using this ontology. Key concepts here are the functional goal, quality goal, function, function constraint, content goal and state constraint. A functional goal for example is a requirement that is fulfilled through a function. Further specifications for the functional goal are the situation, event, effect and subject of the functional goal. The function constraint constrains the situation of the functional goal. A functional and quality goal for example is 'the system shall collect real-time information'.

2.1.4 Context Modeling

Context is defined by [Be18] as: 'in general, a context is defined to be the circumstances that form the setting for an event, statement, process, or idea, and in terms of which the event, statement, process or idea can be better understood and assessed.' Something can be true in one context and utterly fallacious in another context. There are multiple factors contributing to context, e.g. events, intentions, processes, history, location, circumstances, background knowledge, state of affairs, state of a person or device, interactions and many more. The notion of context is gaining more importance in the semantic web community and was the topic of the 2018 Ontology Summit. Modeling context is a challenge because:

- The number of causes of any event is infinite
- There are multi-million of factors contributing to context
- Context is often implicit and enfolded (aka David Bohm's enfolded universe and implicate order [Boh02])
- Intentions of subjects are subjective and hidden

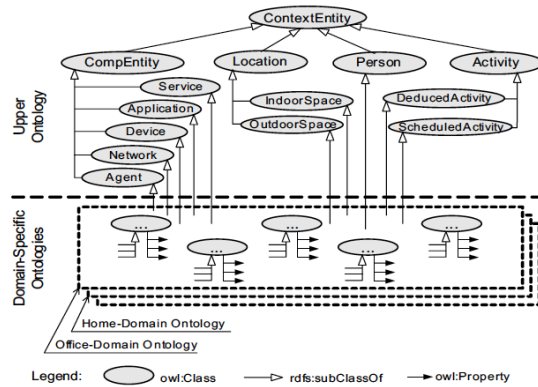


Figure 2.3: CONON Context Ontology [Wan+04]

From one of the author's papers that included context modelling: "[Wan+04] proposes CONON (CONTEXT ONtology), a first context ontology, which models basic units like person, location, activity, and computational entity to specify context. It furthermore consists of specific domain ontologies like smart home and smart office that become integrated in that ontology. CONON can be seen as a top level ontology (TLO) which is one way to model context knowledge because they usually use a very broad terminology.

CONON uses user-defined and ontology defined reasoning capabilities to deduce implicit context." [Waw+] E.g. defining a transitive property 'located in' for a home containing a bedroom and a bedroom containing a person and an inverse property 'contains' to deduce the implicit context that the home also contains the person. Which is knowledge that was not stated explicitly, but is deduced from rules. Other defined rules state properties about the person, e.g. that they are sleeping when they are located in the bedroom, the light level is low and the drapes are closed.

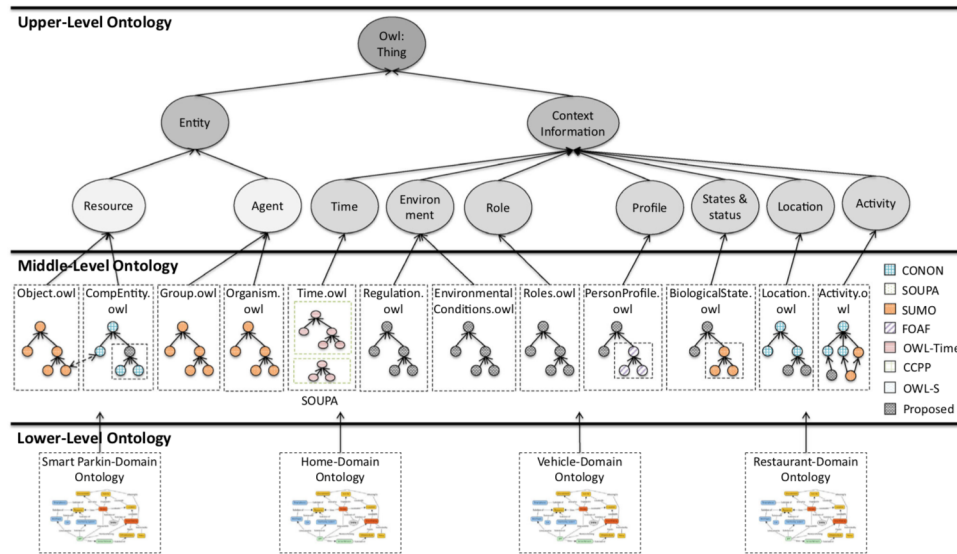


Figure 2.4: 3LConOnt Context Ontology [CFM19]

"CaCont [Xu+13] is a more recent context ontology which builds on CONON. It has a more refined User Model, Device Model, Service Model, Space Model and Environment Model and has similar reasoning capabilities." [Waw+] Defining context also plays a crucial role in data integration, interoperability, natural language, big knowledge and has domain specific drivers. "[CFM19] is a more elaborate and comprehensive work, which takes the concept further. It presents 3LConOnt, a three-level ontology for context modelling in context aware computing. It is structured in top-level, middle-level and lower-level and reuses a manifold of ontologies, ranging from regulations, environmental conditions to roles, objects, activities and resources. 13 context-aware scenarios are described which could be instantiated with this ontology." [Waw+] Figure 2.4 shows the structure and hierarchy of the ontology. The authors considered 64 relevant ontologies for reuse and analyzed them. Some of them were CONON [Wan+04], SOUPA [Che+04], SUMO [NP01], OpenCyc [CCB06], FOAF⁹, CPP¹⁰, OWL-Time¹¹ and OWL-S¹². Most

⁹<http://xmlns.com/foaf/spec>

¹⁰<http://www.w3.org/TR/CCPP-struct-vocab2/>

¹¹<http://www.w3.org/TR/owl-time>.

¹²<http://www.w3.org/Submission/OWL-S>.

of the rest of the paper then shows the instantiations and applications of their models.

2.1.5 IoT Ontologies

This section gives a short overview of IoT ontologies for those not familiar with the developments for the purpose of reference.

1) W3C Semantic Sensor Network (SSN) Ontology

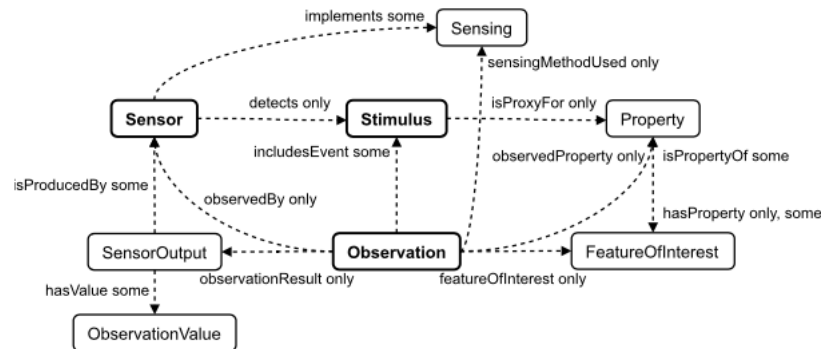


Figure 2.5: Stimulus-Sensor-Observation Pattern [Com+12]

As stated in the paper of the author about digital twins: "*Semantification of sensor data reaches back to the first milestone achieved around 2012, namely the Semantic Sensor Network (SSN) ontology*¹³ [*Hal+18*], created by the W3C Semantic Sensor Network Incubator group [*Com+12*]. SSN described for the first time systematically concepts like sensor, actuator, sample, device, feature of interest, properties and observation and its relationships. Particularly it connects the domains of system, deployment, system capabilities, properties and procedures into one ontology to describe the domain of sensor networks. It is a successful example for the utilization of ontologies, and it was already suggested by the ontologist Barry Smith¹⁴ that it could benefit from the utilization of top-level ontologies and their philosophical distinctions."

 [SGW21]

Figure 2.5 shows an excerpt that relates Sensor, Stimulus, Observation, Property, Sensing and other concepts.

2) ETSI Smart Applications REference (SAREF) ontology

As stated in the paper of the author about digital twins again: "*SAREF (Smart Applications REFerence ontology) [Pov18] is another ontology standard in the IoT ecosystem. It contributes concepts like function, service, command, state, power, energy and task to efficiently deal with IoT devices and their capabilities. It was supposed to enable integrating the domains of healthcare, energy, agriculture, transport, environment, building*

¹³<http://purl.oclc.org/NET/ssnx/ssn>

¹⁴<http://ontology.buffalo.edu/smith/>

and other domains to the IoT." [SGW21]

3) VICINITY core ontology

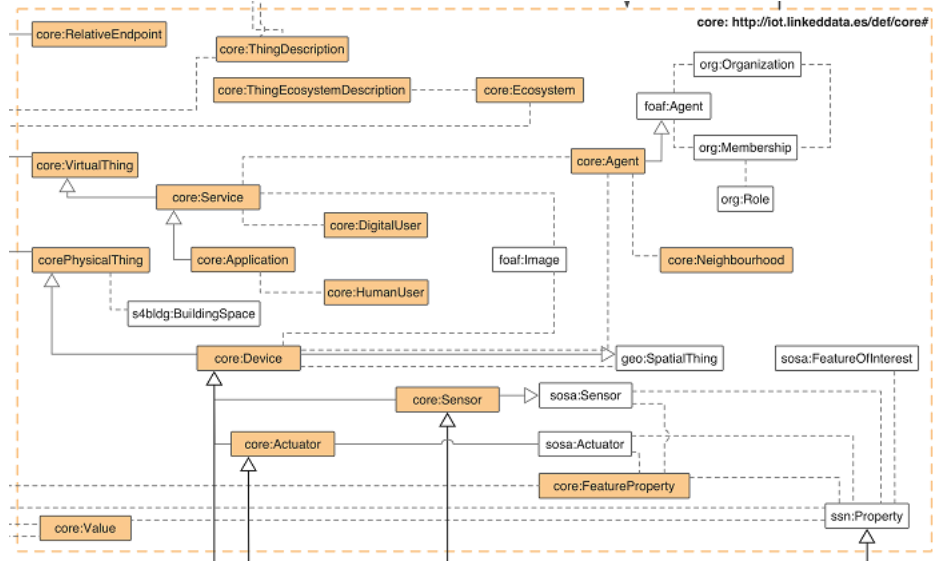


Figure 2.6: VICINITY Core Ontology

VICINITY¹⁵ was one of the Horizon2020 EU funded projects for the interoperability of the internet of things¹⁶. It is an open virtual neighbourhood network to connect IoT infrastructures and smart objects. In deliverable 2.2 the semantic model was developed under lead of UPM¹⁷. It resuses standards such as the SSN, SAREF, W3C Web of Things¹⁸, oneM2M Base ontology¹⁹, FOAF, the organization ontology²⁰, SKOS²¹, Basic Geo Vocabulary²² and others. With a comprehensive methodology requirements were elucidated and the ontology developed. The result was a network of ontologies²³, containing the VICINITY core model as seen in figure 2.6 next to other models.

¹⁵<https://www.vicinity2020.eu/vicinity/>

¹⁶<https://ec.europa.eu/digital-single-market/en/research-innovation-iot>

¹⁷<http://www.upm.es/internacional>

¹⁸<https://www.w3.org/WoT/>

¹⁹<https://www.onem2m.org/technical/onem2m-ontologies>

²⁰<https://www.w3.org/TR/vocab-org/>

²¹<https://www.w3.org/TR/2008/WD-skos-reference-20080829/skos.html>

²²<https://www.w3.org/2003/01/geo/>

²³<http://vicinity.iot.linkeddata.es/vicinity/>

2.2 GENIAL! related Work

2.2.1 SysML in Systems Engineering and SystemC

From the paper "Modeling and Simulation of Cyber-Physical Systems with SICYPHOS" by the author [Waw+15]: *"Complexity of Cyber-Physical System development is addressed by Model-Based Design. In Model-Based Design, virtual prototypes enable the evaluation of the system even if a physical implementation is not available. A valid virtual prototype (resp. model) reduces considerably the risks and pitfalls in system design. Models can also be reused and refined over time, improving their reliability. These characteristics become essential in Cyber-Physical System development, where a large number of devices and complex scenarios have to be validated."*

However, modeling of Cyber-Physical Systems is a challenge [DLV12]. Their heterogeneity requires the use of different, domain-specific languages that must interact concurrently in order to accurately reproduce the system behavior. However there is a lack of languages that are well-suited to model systems across fundamentally different domains.

Modeling and simulation. SysML provides a language to create high-level system models that may cover different domains. However, although it provides a set of diagrams and stereotypes to create highlevel models, there is a lack of semantics on how to use those stereotypes, which usually results in ambiguity when translating those models into domain-specific and executable models.

Co-simulation of different domain-specific tools is possible using tools such as Ptolemy II [BLT10]. However, the acceptance of languages and tools has been very variable depending on the modeling domain, and there are many different languages that are settled de facto standards. The adoption of new tools and languages is not always feasible. Furthermore, industry usually has comprehensive libraries of pre-existing models that have already been tested and are available for reuse. This is the motivation for new standards to reuse models regardless of their modeling language or simulation core, such as the Functional Mock-up Interface (FMI) Standard [Blo+11], which provides interfaces for model reuse and co-simulation.

The SICYPHOS framework presented in this paper proposes a SysML based top-level Cyber-Physical System modeling approach. This model is translated to domainspecific modeling languages: SystemC for hardware/software simulation, SystemC (Wireless) TLM extensions to model network and propagation, and SystemC AMS or Modelica to model analog and physical processes [Mol+14].

Code-generation. A basic work towards generating SystemC models from SysML models is presented in [Bra11]. In that paper, Blocks, Flow Ports, and Operations from SysML are translated into SystemC Modules, Ports and Processes. After the model is set up, it is exported as an XMI file. A translation tool extracts the relevant data from the MOF meta-model and stores this information in C# classes. Operations are associated with state machines or existing code and after two intermediary steps (XML+XSLT) SystemC Code is generated.

A profile for detailed modeling of SystemC TLM with stereotypes is presented in [JKR11]. Stereotypes exist for each TLM modeling construct as well as for SystemC modules and

methods. The motivation of the work is to provide early consistency checking while modeling but not in the simulation. It shall reduce the overall debugging time and effort by using TLM 2.0 rules that can be statically checked as constraints. An example for an informal rule that is realized is 'Initiator cannot realize b_transport method'. Rules are formalized with OCL [Sta14] and are constructed from the SysML meta-model. OCL is a language that enables expressing additional constraints that are difficult to express in UML. In combination with stereotypes (e.g. «tlm_target_socket») the constraints are expressed. After the profile is applied to a specific implementation, a SysML tool validates the model, saves it, and transforms it to SystemC Code via XSL Stylesheets. Supported are structural and behavioral diagrams.

An approach to model SystemC-AMS with concrete semantics is described in [Caf+13]. In SystemC-AMS there are several Models of Computation (MoC), which can be coupled and used to create co-simulations of different domains and applications. In this paper each block or diagram can be assigned with a specific constraint that denotes its MoC or behavior (semantics in brackets): continuous time ('use CT'), discrete time ('use DE') and state machines ('use FSM'). Furthermore, annotations of adaptor-syntax in the form of comments are assigned to ports in order to achieve a precise simulation. To achieve the efforts, a two-phase approach is applied. First the SysML meta-model is transformed into a SystemC-AMS meta-model with the Atlas Transformation Language. In a second step, the actual code is generated from the SystemC-AMS model with the Acceleo code generator.

The existing approaches introduced modeling with SysML and code generations to SystemC/TLM/AMS. With SICYPHOS we combine and extend the different approaches for generating domain-specific code from [JKR11] [Bra11] [Caf+13] and focus on generating the interfaces between different domain-specific languages, and on integrating pre-existing components."

2.2.2 SysML vs. OWL

From the whitepaper "Semantic Technologies - A Comparison" by the author: "Some work has already been done and proposed in this area. There are some approaches that focus on code generations, but also these approaches usually point out some similarities between those languages and how some constructs are directly generated from one language to another whereas others are not. Other approaches merely compare the languages, down to their meta level. Feldmann et. al. in [FKV14] present an approach to measure change influences with compatibility rules. They combine SysML and OWL in a manufacturing use case using plant models. They utilize SPARQL queries to analyze the model for compatibility of ports, data types and function and if corresponding modules are able to work with each other. They accomplish this utilizing the OWL models.

A. Language Comparison

Graves gives a simple comparison of SysML and OWL in his paper [Gra09]. He describes abstract as well as concrete block diagram semantics and how they relate to each other. He states how a simple block diagram can be represented in OWL and translated into its SysML constructs. A more detailed comparison of UML and OWL was done by

	UML Lite	UML Full	OWL
Object-centered	✓	✓	✓
Open-world assumption (not CWA)	×	×	✓
Global scope properties	×	×	✓
First-class status of Properties	✓	✓	✓
Synonyms (not UNA)	×	×	✓
Metaclasses	✓*	✓*	✓
Sufficient conditions	×	✓	✓
Universal concept	×	✓*	✓

Figure 2.7: Comparison between UML and OWL [KA08b]

Kiko and Atkinson in [KA08b], which is the most significant work in this area and closely related to this work. Kiko and Atkinson provide a detailed comparison and also state a lot of differences between UML and OWL though they agree that both languages are essentially the same. That they are essentially the same is disputed by this work, however there are similarities this work's focus in section III is clearly to highlight their differences. Also, their comparison is between UML and OWL whereas this work's comparison is between SysML and OWL. Main interpretations and properties that they compare are summarized in figure 2.7. The table already shows that there are more subtle and gross differences between both languages though there are similarities. In addition, they explain and compare all the elements of UML with the metamodel of OWL in detail and are thus more comprehensive than this work. However, in this work, aspects are addressed that are missing in their work.

B. Code-generation

Olszewska [Isa15] generates OWL ontologies from UML activity diagrams with dynamic processes and validated her approach in a case study of a publication repository domain. She presented a simple User Login into a System which activities get translated into OWL data and object properties. NASA's Jet Propulsion Laboratory and Systems and Software Division department generates its ontologies from SysML diagrams. In this paper [Jen12] they describe that embedding classes is straightforward whereas embedding object properties is more complex and describe some transformations and what they transformed. They also state that they test for consistency, well-formedness and satisfiability. For their transformation into OWL, they use an operational Query/View/Transform transformation."

Some research was done in this area in order to help facilitating the understanding of both technologies for people struggling with the intricacies of semantic technologies. Though SysML uses semantics there are considerable differences between both technologies which have ripple effects on paradigm, focus, data format, how things are modeled

and so on. To state it clearly ontologies and SysML are not nearly the same. Ontologies originate from the field of artificial intelligence and SysML from systems modeling. In the following there is a summary of some, but not all of their differences (from the authors whitepaper "Semantic Technologies - A Comparison" [Waw17]):

"1) Resource Identifier (Unique Resource Identifier vs. regular id)

In OWL we see the Unique Resource Identifier that is an http Website link which connects all items in the ontology and makes them not only unique but available to find on the internet. This is in contrast to SysML where there's merely a simple id with usually just some anonymous number combination. This makes it unique, but is rather not handy when trying to reuse it or find it or grasp the meaning of the modeled entity.

2) Paradigm (IoT vs. Systems Engineering)

In systems engineering we still see a System Integrator and project leader who manages the system overall, integrates new parts and ideas, has system management tasks as well as overview of the system. In the IoT paradigm this system integrator is nonexistent since input comes from a variety of contributors, parties and vendors. Ontologies can clearly be placed in the more encompassing IoT paradigm since it is possible to represent most things semantically. Everything that can be put into words directly has a meaning and all of systems engineering is subsiding into the IoT paradigm by virtue of being a subclass of it. System Engineering entails organizational structures, processes, workflows, business models, requirements engineering, simulation models and more. And even those systems are now being enhanced into social-cultural systems by the INCOSE group. So the boundaries between systems engineering and IoT are also blurring like in many other disciplines. Additional note: Ontologies are not restricted to the IoT, but originate from the semantic web.

3) Focus (Interoperability vs. Modeling)

With ontologies the focus is on interoperability, which means providing the semantics to enable two technical systems to understand each other and exchange information based on a semantic standard. In SysML this is not even slightly the case, there is no emphasis on interoperability at all since the models are used in quite different ways. Or at least names are chosen in a way that they remain in the same data silo they are supposed to serve – a technical specification within a systems engineering project. In SysML, a modeler usually designs one single overall system which entails all the components of the system and which is not used by another system or which information is not used by another system. Ontologies map the information of each system in a way that it is accessible from and can also be written to - all other parties. In SysML this is not wanted since the specification is usually preferred to come from the system integrator and is fixed through some forms of the requirements. In other words instances cannot be populated by third parties, but only by the designer within the SysML tool.

4) Data (queriable structure vs. non-queriable data format)

Ontologies come from a language that can in some aspects be compared to a database. In fact OWL is already quite close to what is known today as relational databases. One such characteristic is that an OWL file can be queried via an SPARQL endpoint. This leads to autonomous agents who independently read data from ontologies to perform their duties

and tasks and then write back to them. Examples are automobiles, drones and so on that read semantic information from independent sources in navigation units or other areas of application. In SysML such a query is absent. The model has a certain structure that can be read, but which is rather fixed, not extensible, and not modifiable, except from the modeler. It can be used to generate code in different languages.

5) *Exchange (global sharing approach vs. limited exchange format)*

Ontologies, also based on their paradigm, which was mentioned above, are based on a fundamentally different approach to exchange their data than in SysML. It already starts with how the vocabulary is built and used. In ontologies great care is put into naming and a lot of characteristics and properties have to be applied to naming. One is generality for example, which is often used in upper ontologies that many categories fit into such an ontology. The naming must be general as to be compatible with many approaches and understood by many parties and vendors. Again, in SysML files and models of a project are usually just used within the project and do not require proper general naming as with ontologies as the exchange is limited by its use. Ontologies are based on the Open Linked Data paradigm and the data is exposed. In the future we will see more general approaches also containing general function on top of that vocabulary. These are important that smart systems and autonomous agents are able to make use of the data and use it in a variety of contexts.

6) *Knowledge vs. System Information (Modeling of meaning and complete information vs. modeling of system properties and parameters)*

Ontologies allow for the modeling of true knowledge, which means that also a classification and reuse of concepts and ideas can be made. To stay with a simple example from the well-known pizza finder tutorial²⁴ it is possible to model the concepts hot topping or cheesy topping on top of the classical pizza type and categories. This makes it possible to search for all pizzas that are hot or are cheesy simply by modeling the categorization on top of the concepts. In SysML such a modeling approach is not feasible. First of all the modeling modalities are missing. In SysML it is possible to create stereotypes with similar characteristics, however at least reuse is not ensured like when using ontologies. A SysML block has operations and properties, however in ontologies an arbitrary chain of relations can be created and linked to various classes.

7) *Modular (exchangable and extensible vs. rather rigid for one system)*

Ontologies are modular, they can be imported into each other and easily linked which is their purpose to be extensible, distributed and combined. SysML is quite rigid in this regard.

8) *Make-up (non-graphical vs. graphical)*

Another more fine grained difference is that SysML is a graphical language for documentation where the system is visualized depending on the tool used. Ontologies are also visualized for viewing within tools like Protegé for example, but also offer to be visualized on demand with other visualization ontologies for example. In this regard an ontology is more flexible.

9) *Ontologies are tool independent, SysML is not*

²⁴<http://owl.cs.manchester.ac.uk/research/co-ode/pizza-finder/>

As mentioned above SysML is visualized depending on a tool. And tool and language are usually deeply intertwined. So much so, that the model cannot be used without the tool and some efforts have been made to solve this issue (e.g. Modelbus). Ontologies are usually being manipulated via the OWL API and are thus being able to be used within any java program. However other implementations like the C++ API seem rather basic and unusable.

10) Smart knowledge base vs. unintelligent rigid models)

In ontologies we have something called Inference, which allows for smarter knowledge organization. For example we can create equivalent classes, which are classes with certain properties and conditions. When other classes meet these conditions the equivalent class gets inferred to be a subclass of the other class, which has these conditions. In this way it is for example (to stay with the pizza examples) possible to categorize pizzas with special toppings or characteristics as part of the class InterestingPizza, a feature, which cannot be modeled in SysML.

Additional note: Furthermore in ontologies it is possible to organize the knowledge with transitive and inverse object properties to make implicit knowledge explicit

11) SysML is more dynamic

SysML is more dynamic in respect to its activity and state diagrams as well as operations which are not – or not yet well represented in OWL."

Note: It should be possible to convert such diagram fully into OWL.

2.2.3 Constrained-based Configuration in Expert Systems

From the paper "A Knowledge-based Approach for Engineering 4.0" by the author [WG18]: "Danninger [Dan15] states that most complex software systems are still generally configured by hand and by human experts. This process is error-prone and tedious and it is hard to consider all applicable constraints and interdependencies. He states that constraint programming with constraint solvers is supposed to be a promising field of research and are able to handle the complexity. He gives an introduction of constraint programming and a simple example with an overview of some current implementations. For him the challenge is not in the solving process itself, but in providing users with appropriate and intuitive ways to specify constraints, which is also part of this work in IV with the creation of a graphical, easy to use user interface. He mentions ConfSolve, an object-oriented configuration language that generates its final configuration in the form of an object tree. And other approaches that contain self-healing modalities, which re-configure the system when the configuration changes. Furthermore, he mentions Fresh [WS08], a model driven engineering tool that transforms feature models into constraint satisfaction problems. Fresh supports constraints between features, which also constrain the number of components. The feature models are then reduced to constraint satisfaction problems (CSPs) and solved. Though Danninger thinks of this research as promising, he still criticizes that there are few cited industrial applications, that most still require a lot of manual configuration and that capturing complex domain constraints and translating them to declarative models is difficult, which was also discovered in this work.

Another constraint configuration tool is KONWERK, mentioned in [Bir96]. It is supposed

to support configuration and design tasks in technical domains. KONWERK consists of four modules that cover the tasks of representation of domain objects, representation and processing of relations, constraints and heuristics, formulation of the configuration task and control of the configuration process. KONWERK is explained in [Bir96] according to the Nitra River Case Example: A river with its tributaries along with monitoring, emission and other kind of nodes and points. At the emission points certain treatments are implemented in order to reduce the pollution of the system. There are six total objectives like minimizing cost or maximizing the concentration of e.g. oxygen, which are complementary or in opposition with each other. KONWERK is not only restricted to technical domains, but was also applied to layout design or environmental projects. A Basic Optimization module translates the knowledge base into constraints so it can be used and solved by a constraint solver. The knowledge base of KONWERK defines the knowledge in a declarative way and has the types hierarchy of concepts, compositional hierarchies, conceptual constraints, conceptual objectives, optimization tasks and strategies. The tool presented in this paper is similar to KONWERK in its description of domains and constraints, however more advanced in respect to its graphical user interface and its consideration of time constraints and predictive possibilities.

Waldschmidt et. al in [Pet96] describe a system-level construction methodology which is used for the synthesis of analog and digital systems. They describe how expert systems can be used to define the task of a partitioning problem through conception hierarchies and support the distribution of limited design resources through constraint nets. They use it to make important design decisions at system level like for example the setting of bit widths. Their construction process consists of the following four constructions steps: splitting objects and integrating objects via has-parts relationships, specialize objects via is-a relationships and parameterize objects. It is a construction process which is followed by the tool KANDIS and also show similarities to KONWERK. KANDIS takes as input a VHDL hybrid description translates it into a graph-based intermediate format, which is then again translated into instances and concept hierarchies on the knowledge-base level. KANDIS is unique in the way that it works on construction at system level. As part of the partitioning process the tool automatically generates a corresponding converter. During the design and construction process, the different blocks require different resources, which continually reduce the ranges of their design parameters and are handled by constraint nets. Examples of these parameters are: power consumption, delays, data rates, sampling frequencies and others.

In [GCS90] Günter et. al. present a way to separate control knowledge from structural knowledge in order to circumvent the disadvantages of those expert systems, namely knowledge acquisition, consistency maintenance and modification of the knowledge base. Here rules are taken out of the knowledge base, though the structural knowledge is designed in a way that it supports a control mechanism. In [GCS90] the construction process consists of an agenda, the selecting from the agenda, value determination methods, controlling constraint propagation, conflict resolution knowledge, phases and strategies and control rules. The agenda are the executable construction steps from the partial solution, the comparison of concepts and their instances, namely specialization, decomposition,

aggregation and parameterization. Selecting from the agenda is done by agenda selection criteria, which yield preferences of criteria to select. If the selection is ambiguous, value determination methods help in the selection of criteria e.g. by asking the user or evaluating a function. What is done in this paper by the constraint solver Z3, is integrated here: controlling the constraint propagation process. Furthermore, rules are implemented that are used to navigate in case of a conflict. Strategies are packed up control knowledge, which consists of a combination of aforementioned tasks and steps. Control rules are rules that determine how to select and terminate construction phases; it helps select a strategy.

ENGCON described in [Run06] is a constrained- and structure-based configuration tool, which uses functional and predicate constraints to describe dependencies between concepts of the knowledge base. An external constraint solver solves the constraints.

Kühn in his paper [Küh01] compares different approaches of configuring with knowledge-based systems. He differentiates between rule-based, structure-based, constraint-based, resource-based and case-based systems, which apply best depending on the kind of knowledge used and the configuration task. He especially describes how to integrate behavior-based knowledge with aforementioned systems to aid with problem solving. To enhance constraint-based systems with behavior, state transitions, the exiting of a state and the entrance of a state can be added with constraints for requiring or forbidding that state. The passive role of the constraints is to check consistency and the active role to constrain the potential behavior of one or more parts of configuration. He concludes that behavioral knowledge is best combined with structure-based and constraint-based systems. In this paper the structure-based as well as constraint-based system are proposed and implemented."

2.2.4 Self-Aware Cyber-Physical Systems

The term self-aware cyber-physical systems (CPS) has some close relationship to the IoT and to autonomous agents, though it has also its own focus. According to [JDR17] there is a hierarchy of self-* properties comprising self-awareness, context awareness, self-configuration, self-healing, self-optimization, self-protecting and self-adaptiveness. In [Gur+13] there are even more properties mentioned like self-organization, self-discovery, self-description and self-energy-supplying. This field combines findings from software engineering, biology, artificial intelligence, control theory and embedded systems. It is estimated that self-awareness properties will be part of any software module in the future and have some form of embedded intelligence. These features will have to be considered in the early development stages of CPS.

Considering System-on-Chip technologies there is also an increasing demand for self-awareness coupled with hardware attributes like performance, robustness, energy and resiliency [DJS15]. State of the art in this research area is the exploration of what self-awareness is and how to employ the self-* properties and even let them work together to achieve higher-level goals.

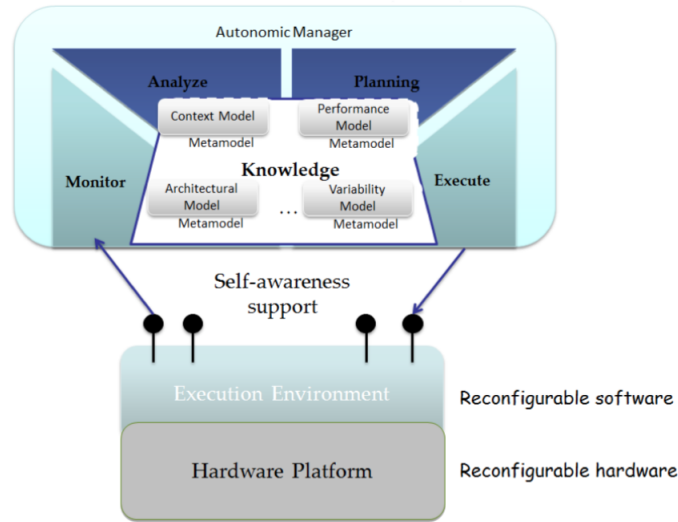


Figure 2.8: MAPE-K Architecture [Gur+13]

These self-* properties are realized via the MAPE-K architecture, which stands for **M**onitor, **A**nalyze, **P**lan, **E**xecute and **K**nowledge, shown in figure 2.8. The monitor phase basically records sensor information from a variety of sensors like light, temperature, pressure, weight, gyro sensors, input pins, cameras and so on. The analyze phase processes and analyzes this information and the execute phase puts actions on physical entities by actuators (e.g. activating roller shutter etc.) The planning phase decides on which actions to take and is realized via event condition rules, objective functions and prediction models (bayesian networks, decision trees [WR14], fuzzy logic). These features bring in new challenges like prediction models at run-time, real-time conflict management between rules, and learning from experience with case-based reasoning [Gur+13]. This work is significant in the context of knowledge representation as these systems will all work based on a knowledge base and process and query semantic information, which will be the basis for understanding information and making meaningful decisions.

Figure 2.9²⁵ shows the development from a regular and basic cyber-physical system (simple reflex agent) to an advanced learning agent (general learning agent). In the basic system, there is a simple control loop from measuring the environment through sensors to taking actions on the perceived values through conditional if-then statements and then actuating on the environment.

Then, step by step, more complexity is added. The goal-based agent already has awareness about its state, it's goals and how to achieve these goals by knowing what his actions do.

The general learning agent is beyond the scope of this work, but is mentioned for completeness as it also interacts with a knowledge base in combination with machine learning techniques. Types of machine learning covered by this agent are:

²⁵https://en.wikipedia.org/wiki/Intelligent_agent

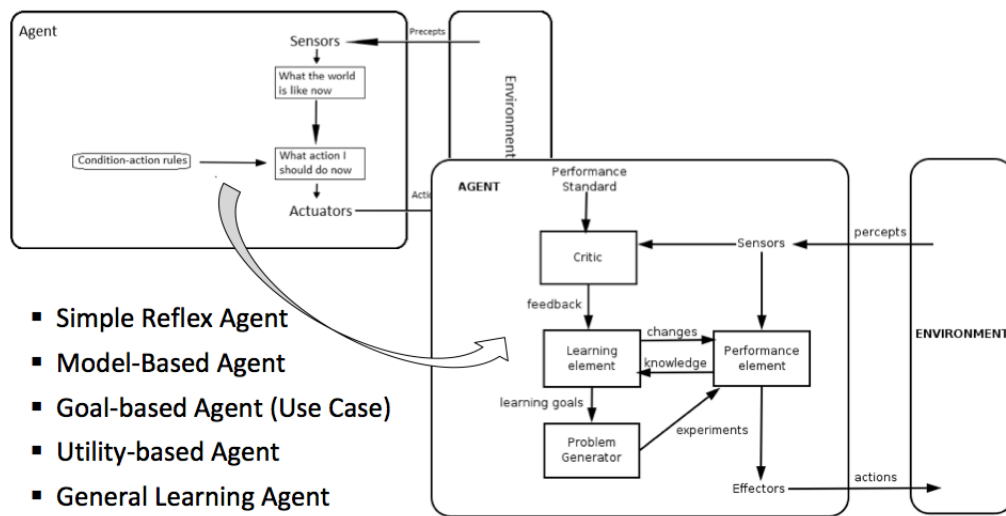


Figure 2.9: Autonomous Agents Development [Wik19]

- Supervised learning (e.g. function approximation, prediction, pattern classification, natural language processing, image classification and segmentation, object detection and localization)
- Unsupervised learning
- Reinforcement learning (e.g. game theory)

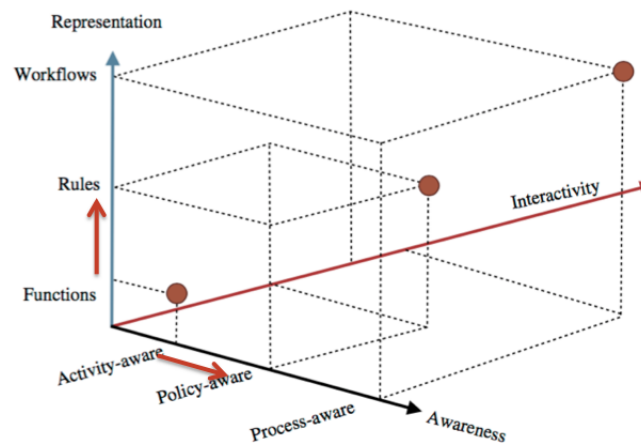


Figure 2.10: Autonomous Agents Vision ([Ver+09], modified)

The general learning agent has a critic, which helps to evaluate the agent's behaviour based on an external behavioural measure. The problem generator suggests exploratory

actions that lead the agent to new experiences. The learning agent carries out improvements by utilizing knowledge and feedback. The performance element processes sensor information and acts on it. It corresponds to the simple reflex agent from basic cyber-physical systems.

In figure 2.10 it can be seen that technical systems are not just a collection of functions and activities, but there are actual levels to representation and awareness, which become more powerful as their degree of freedom advances. The most adept autonomous agents are those that collaborate in workflows together and are aware of the whole process that is going on. As their power increases usually their level of interaction does as well. This corresponds with a powerful vision of these systems, as by being highly aware they will be able to achieve tasks that were originally reserved to humans.

Chapter 3

Foundations for Knowledge Representation

The information found in this chapter is not the authors original work, but refers to other works, mostly from Prof. Giancarlo Guizzardi, Dr. Nicola Guarino (who I both met at conferences), Prof. Maria C. Keet, Prof. Barry Smith and work from UPM.

3.1 Basic Foundations for Knowledge Representation

3.1.1 From Thesauri to Common Logic

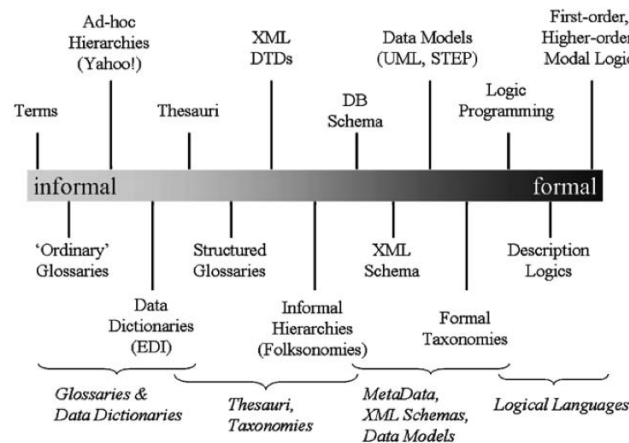


Figure 3.1: Continuum of Expressiveness and Formality [UG04]

When it comes to expressiveness and formality, we can speak of a continuum. On the left side of the continuum are terms and sentences in prose or a plain text document. Though very expressive, this is basically not understandable to a computer, except par-

tially by using e.g. statistics or string manipulations. Figure 3.1 shows an outline, which is explained in the following with some additions made.

Terms - a word or phrase used to describe a thing or to express a concept, especially in a particular kind of language or branch of study.¹

Ordinary Glossary - is a list of words with their meaning. Often in alphabetical order about a specific text or subject.

Ad-Hoc Hierarchies - *"Ad hoc hierarchies such as Yahoo! are sets of terms with a relationship between terms, but where no formal semantics for that relationship is defined."* [And07].

Data Dictionary (EDI) - *"are more formal models of information, often of relational databases, where each term and relation has an associated natural language definition (EDI stands for standardized Electronic Data Interchange)."* [And07]

Thesaurus - *"is a word list that is not ordered alphabetically but according to conceptual relations."* [And07].

Structured Glossary - *"may include additional relationships among the terms in the glossary."* [And07].

XML DTD's - *"are Document Type Definitions in eXtensible Markup Language, used for communication among software systems. XML supports nested, or hierarchical information structures, but is a language for defining syntax that has no associated constraints on semantics."* [And07].

Informal Hierarchies (Folksonomies) - Also known as collaborative tagging or social tagging. Are basically tags done by end users. They have the advantage that they are flexible and based on the vocabulary of the users. They are not a shared common vocabulary, though they may be shared between different participants. They are not a taxonomy, as they do not have a hierarchy, and they are not well-defined classes. Though not formal, they may have informal relationships.

DB Schema - There are SQL, NoSQL, relational, graph databases and more. *"DB Schemas are Data Base structures that have more formal definitions of the meaning of terms and relations, usually by employing statements in a database constraint language."* [And07].

XML Schema - XML is structured data. It does not yield any understandings, in contrast to ontologies. However, XML is the basis of OWL and can be used for machine readability and to send simple commands, properties, terms or text between devices.

JSON - JavaScript Object Notation is slowly substituting XML because it is more lightweight and usable for data-interchange.

Data Models (UML, STEP) - *"couple taxonomies and defined relationships with a semantics for representing process and action. UML, the Unified Modeling Language for specifying the design of object-oriented systems [...] exemplify this kind of information model."* [And07]

Formal Taxonomies - basically, taxonomies consist of formal subClassOf / superClassOf hierarchies. Additionally, there are sometimes a few basic relationships present, like 'narrowerThan' or 'widerThan' or 'similarTo'. Usually ontologies on the other hand have

¹<https://www.lexico.com/en/definition/term>

more rich object properties and data properties and have restrictions. A special form of the taxonomy is the **partonomy**, which enhances the taxonomy with the hasParts / partOf relationship.

Frames - "include a range of standard AI languages that have terms, relations, and inheritance of properties. Examples are the Open Knowledge Base Connectivity (OKBC) protocol and the ontology editor and knowledge acquisition system Protégé." [And07]

Logic Programming - Similar to programming languages logic programming expresses logic in the form of facts and rules.

Object Constraint Language (OCL) - allows constraining UML models. E.g. values of properties or cardinality of relationships.

Topic Maps - Topic Maps are similar to RDF. They are serialized in XML and are an ISO standard. They have *topics*, *association* and *occurrences* as elements, which basically form a graph with edges and nodes. They do not have restrictions as ontologies have. **Concept Maps** are similar but not an ISO standard and more of a diagram.

Description Logic - Description logic is the basis of OWL. It works so well, because it is a decidable language, that yields results with reasoning. It consists of classes, instances and properties and logic constructs. It is partitioned in TBox, which states general statements about the world with its terminologies and the ABox, which makes assertions with concrete instances and their relationships.

First Order Logic, Higher Order Logic and Modal Logic - FOL allows expressing logical sentences with a variable and quantifiers. It has syntax, semantics and rules. In comparison to propositional logic, FOL allows predicates. Higher order logic is more expressive than FOL but less well behaved. Modal logic allows to qualify statements, e.g., 'usually', 'necessarily', 'possibly'.

Common Logic - CL is a first order logic family language, which is most expressive and thus all the semantic web languages can map into that language. Thus it is also not made for computation, is not decidable and useful when it comes to reasoning and inference. It can be represented in the Common Logic Interchange Format (CLIF). Its goals are according to [Kee18]: "1) *Common interlingua for a variety of KR notations*; 2) *Syntactically as unconstrained as possible*; 3) *Semantically as simple and conventional as possible*; 4) *Full first-order logic with equality; at least*; 5) *web-savvy, up-to-date*; 6) *Historical origins in Knowledge Interchange Format (KIF)*."

3.1.2 From Reality to Ontology

An ontology never captures reality itself, but rather a conceptualization of it that was first filtered through our perception. When we select a language, e.g. English, each term/expression (e.g. the word 'Person' or the word 'implements' as used as an object property for example) of that language L commits to a certain conceptualization through an ontological commitment K. That language is itself subject to various interpretations I that yield different models for everybody. The sum of those models is Md (L). A good ontology captures the intended scope of those models and maybe a bit more of them

(slightly over-constrained). A bad ontology does not capture some part of the intended models and may even capture some part of other models. Usually if we define two object properties, they are unconstrained and could be used in different ways. Therefore, according to [GOS09] we then need to make sure that we admit only those models to which our conceptualization commits. Formally making a conceptualization explicit can be made in an *extensional* or *intensional* way. An *extensional* way would be to list possible variations of that conceptualization, give examples and state them explicitly. An *intensional* way would be defining specific axioms of e.g. OWL2 like transitivity, inverse, symmetry, irreflexive etc.

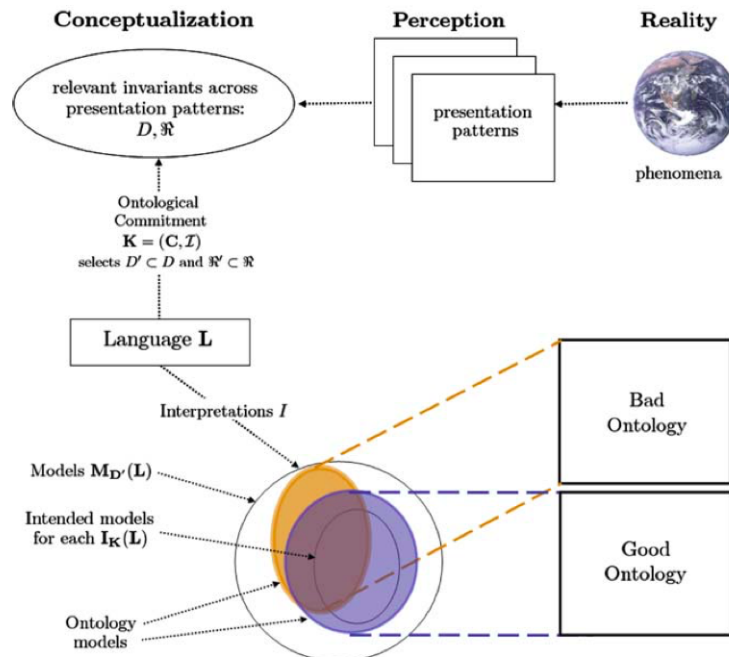


Figure 3.2: Reality and Ontology [GOS09]

3.1.3 Categories of Ontologies and their Meanings

In source² it gives a good definition and explanation of types and is referenced here. Just some additions were made. The source was not referencing task and middle level ontologies. Sometimes other ontology types (data ontology, interface ontology, product and process ontology) are referred, but they can be classified under existing types of ontologies.

- Classification according to Purpose
 - Application Ontology: Used during run time of a specific application implementing an ontology putting constraints on the axiomatization for the

²<http://km.aifb.kit.edu/sites/cos/>



Figure 3.3: Types and Categories of Ontologies

terminological service, i.e., the reasoner. The typical trade-off between expressiveness and decidability requires a limited representation formalism. In a description logics based application this would coincide with the TBox. Application ontologies may also describe specific worlds (semantic descriptions, knowledge base, metadata, semantic metadata or simply instances) In a description logics based application this would coincide with the ABox.

- Reference Ontology: Used during development time of applications for mutual understanding and explanation between (human or artificial) agents belonging to different communities, for establishing consensus in a community that needs to adopt a new term or simply for explaining the meaning of a term to somebody new to the community. Although parts of the reference ontology can be formalized in a TBox as well, description logics are usually not expressive enough for reference purposes.
- Task Ontology: Task ontologies usually play together with domain, application and upper ontologies. They often state a problem context and how to solve those problems in combination with domain knowledge. They can also be very abstract.
- Classification according to Expressiveness
 - Heavyweight Ontology: Heavyweight ontologies are extensively axiomatized and thus represent ontological commitment explicitly. The purpose of the axiomatization is to exclude terminological and conceptual ambiguities, due to unintended interpretations. Every heavyweight ontology can have a lightweight version. Many domain ontologies are heavyweight because they should support heavy reasoning (e.g., for integrating database schemata, or to drive complex corporate applications). As with all dimensions, the borderline between light and heavy weight is not clearly delimited.
 - Lightweight Ontology: Lightweight ontologies are simple taxonomic structures of primitive or composite terms together with associated definitions. They are hardly axiomatized as the intended meaning of the terms used by the commu-

nity is more or less known in advance by all members, and the ontology can be limited to those structural relationships among terms that are considered as relevant.

- Classification according to Specificity
 - Generic Ontology: The concepts defined by this layer are considered to be generic across many fields. Typically, generic ontologies (synonyms are "upper level" or "top-level" ontology) define concepts such as state, event, process, action, component etc.
 - Core Ontology: Core ontologies define concepts which are generic across a set of domains. Therefore, they are situated in between the two extremes of generic and domain ontologies. The borderline between generic and core ontologies is not clearly defined because there is no exhaustive enumeration of fields and their conceptualizations. However, the distinction is intuitively meaningful and useful for building libraries.
 - Middle Level Ontology: Middle level ontologies are an intermediary, which mitigate the abstraction levels given by an upper ontology towards domain ontologies. They provide levels/a taxonomy of refinements/specializations and advantages when it comes to extending knowledge domains and for knowledge management. They may be very similar on the level of abstraction, but not equal to core ontologies.
 - Domain Ontology: Domain ontologies express conceptualizations that are specific for a specific universe of discourse. The concepts in domain ontologies are often defined as specializations of concepts in the generic and core ontologies. The borderline between core and domain ontologies is not clearly defined because core ontologies intend to be generic within a domain. Thus, it is usually hard to make a clear cut between generic and core as well as between core and domain ontologies. A concept such as software component would be placed in a core ontology for application servers for reuse in every possible domain ontology we can think of.

3.1.4 Design Guidelines and Methodologies for Ontologies

This section is research done in combination with a master thesis entitled "Towards a true microelectronic and automotive digital roadmap" [Sun20].

Ontology Methodologies

Uschold and King's method recommends the following four steps to develop an ontology. [CFG03] The steps are

1. *Identify the purpose*
2. *Building Ontology*

- *Ontology capture - Understanding primary concepts and their relationship in the domain*
- *Ontology coding - Representing the concepts in the formal language*
- *Integrating existing Ontology*

3. *Evaluation*

4. *Documentation*

There are approaches to identify the primary concepts to develop an ontology. The approaches are top-down, bottom-up, and middle-out. Top-down - starts generalization of the concepts and specializing the same concepts in a more specific way. The bottom-up approach is the exact opposite of the top-down approach. This approach starts with a specific concept and expanding and generalizing the concepts. Middle-out approach focuses on the key concepts for the ontology and later specialize and generalize into other concepts.

METHONTOLOGY is a framework, which supports developing new ontology or reuse the existing ontology. [CFG03] There are three main processes. Each of these processes have specific activities. The processes are management process, development process, and support process. The activities available for each of these processes are management process is divided into the following activities scheduling, control, and quality assurance. The development process consists of specification, conceptualization, formalization, implementation, and maintenance. The support process carries knowledge acquisition, evaluation, documentation, configuration management, and integration. [PSM08]

Problems and Best practices

The advantages of ontology are designing, sharing, analyzing, reusing the domain knowledge among different people and software tools. There are several interoperability issues in interchanging data among the systems and reusing available information. The classifications of interoperability are:

1. *Technical Interoperability*
2. *Syntactical Interoperability*
3. *Semantic Interoperability*
4. *Organization Interoperability*

In the following section, Semantic Interoperability and their best practices is discussed in detail. Semantic Interoperability mainly deals with the different structure of the ontologies, terms (vocabulary) used and their meaning. Understanding these primary problems encourage us to design and build less problematic ontologies in the future. Implementation of these recommendations will improve the easy reusability of the ontology as well. There are five different areas which provide best practices for semantic interoperability. [GSA15; KP11]

1. *Ontology format - using a consolidated format to describe the ontology. Wrappers and translators are in the place of a different system. Terms used in the ontology. Testing the semantic document should help the interoperability of the ontology.*

2. *Metadata - minimizing the usage of labels and comments, version date, author information in the ontology.*
3. *Quality of the ontology - dereferenceable of the ontology will help to automate. Verifying the rudimentary issues of the ontology can be done with tools*
4. *Reuse the ontology - reusability of the ontology helps in developing the vocabulary of the domain. Proper documentation of the ontology should help to reuse the ontology available.*
5. *Managing Namespace - tools are available to select good namespace for the ontology and namespace helps to automate necessary tasks.*

Using vocabulary is crucial for any domain ontology design. From the most reliable Gene Ontology, there are a few tips and techniques to define vocabulary in a better way. They are, to utilize the relevant terms that are used by prominent researchers and group. Designing ontology involves various sources. To avoid inappropriate usage of the term, the designer should utilize specific terms used by the domain experts. It is also crucial to understand the usage of the terms involved in the design with the help of the experts. Recognize the terms which are overlapping or incompatible with the domain. It is important to crosscheck the terms with the domain and create a list of terms which are problematic for future reference. There are few terms which have different meanings depending on their usage. Relating specific term with the alphanumeric identifier will mitigate this problem. The identifier helps in reusing the existing terms and benefits in updating the ontology with a new version when there are no changes with the terms associated. The term used in the ontology should have unique meaning.

The usage of the good namespace in the ontology will improve a better understanding of the ontology and also improves the readability and reusability. The naming convention of the ontology depends on the software tool used to design the ontology. For instance, case sensitive or not, delimiters that can be used and etc. A few proven recommendations for the namespace is in the following paragraph.

Firstly, the name of the classes and properties should have consistent rules throughout the ontology. The class name should use either uppercase or lowercase. If the name of the class has more than one word, using appropriate delimiters and formats such as camel-case, uppercase for each single word in the same. For instance, the class name is "Living being". This name can be represented as follows "Living_Being" or "Living-Being" or "Living_being". Italics format can be used to represent the class name. Secondly, the class name can use either a singular or plural form. There are different takes on using the singular or plural form of the noun. Since the class expresses the collection of objects, the class name can naturally take the plural form. For example, Attacks is the class name, which represents different security attacks in the system. On the other side, the singular form can be used to define the term. The class does not always represent a group of objects. If the ontology designer uses the plural form, the class name should take plural form consistently throughout the design.

Relation expressions establish the connection between the class and the slots and help to differentiate. The expression should be unambiguous and consistent throughout the ontology. One of the common practices is adding "has - or - of " expression to the slot

name. It is not recommended to use abbreviations or acronyms in the ontology. Ontology users can misinterpret the used abbreviations or acronyms. Avoid adding strings such as “Class_”, “slot_” to the terms. [NM19]

Basic Rules for Designing an Ontology [ASS15]

In this section, we will discuss rudimentary rules and understanding towards designing ontology in a more beneficial way. [Hor11] We have already presented information about the consistent designing and development of an ontology. Realism - Any ontology should represent the reality of the considered domain (with the features and relationship) irrespective of the developers’ perception. Perspectivalism - An ontology should not only help to reach limited goals but also it should support to create theories that are rational, descriptive. Expressing reality with all the features in a single ontology is an intricate task to achieve. The developers can adopt the modular approach to develop ontologies to reduce the information crowding. Fallibilism - The design that expresses the reality can be proved to be false. The features, in reality, are subjected to improvements or changes. In any evolving domain, the improvements or changes to the features are constant. To reflect reality in the ontology, parts of the design should go through the corrections required. Take away for designing ontology is

- Maintain version for each ontology. It helps to keep track of the changed and in correcting the ontology with the required information.

Adequatism - Ontology should be designed and developed with fine details which can help to associate with other domain and reuse the existing design and features. Adequatism increases the interoperability of the ontology as well.

There are other few additional rules to be considered for designing ontology. We have discussed earlier the reusability of the ontology. Reusability enables the ontology developer(user) to rectify the existing problems with the domain and improve the quality and accuracy of the features presented. Though manifesting the reality of the domain is the essential part of ontology design. The ontology should also address the usage of the confined domain. There should be the right amount of balance between reality and real-time usage of the ontology. The development of ontology is a continuous process so the design should help to manage, update, correct, and extend the domain with relevant features. It is possible to expand the ontology with the nearby domain. The starting point of the design of domain ontology should express the features which are vivid and simple to understand the domain. This is known as “the principle of Low-hanging fruit”.

The common pitfalls should be considered carefully in OWL. [Rec+06] They are

1. Usage of the disjoint class
2. Identifying and applying apt restrictions (Existential restrictions).
3. Adapting Open World Assumption
4. Understanding the difference between domain and range.

All the individuals should at least have one relationship with the property. Some restrictions or existential restrictions is the default restrictions in OWL. Open World Assump-

tion (OWA) helps to maintain the correctness and add missing or new information to the ontology.

As mentioned above, the best practices and basic rules which provides complete guidelines to design and develop the ontology in an efficient way.

In the following section, we are going to examine how to develop ontology. There are several methods possible to build or develop ontology. Based on the best practices, we should support the concept of reusing the existing ontologies and sustain consistency throughout the development process of any ontology. Domain Knowledge Acquisition Process (DKAP) methodology assists with the above-mentioned issues. The steps carried out through this methodology is unlike other ontology developing methodologies. The structure of the DKAP methodology as shown in Fig. 3.4 There are nine steps involved in developing ontology. [SF07]

1. Defining domain and scope of the ontology - The crucial initial step provides support to understand the purpose (domain) and context (relationship) of the ontology. Establishing the limit to explore the domain and their associated relationships.
2. Reusing the existing ontology if available - This step shows the major difference from the existing methodologies.
3. Planning the activities for developing the ontology.
4. Acquisition of the data and interpret the gathered data for the chosen domain.
5. With the help of the interpreted data, develop a rough version of the ontology.
6. Test and perform cleanup for the existing data in the rough version of the ontology.
7. On the completion of the ontology development, check the Coherence and correctness of the ontology.
8. If any issues observed previously, this step paves the way to fix them by adding appropriate data to the ontology. During this step, data will be gathered and interpreted again.
9. The final step of the methodology is promulgating the ontology and sharing the experience in developing ontology.

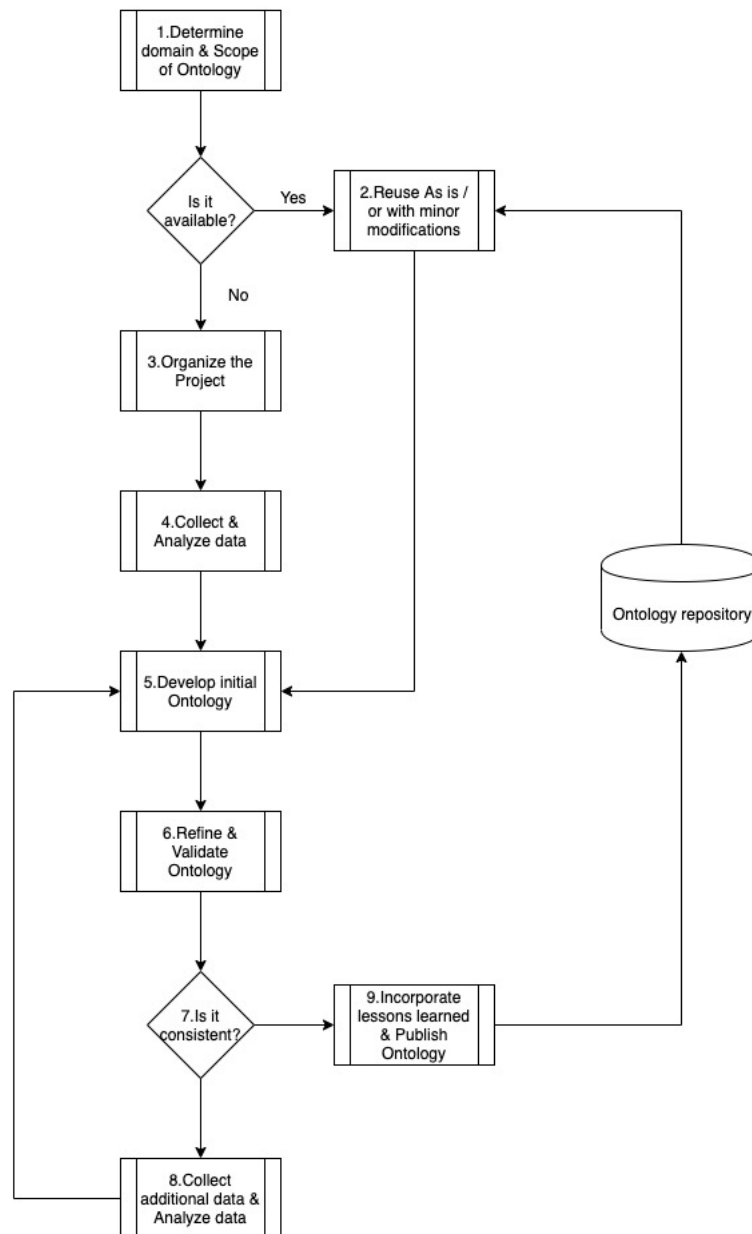


Figure 3.4: DKAP Methodology Structure [SF07]

3.2 Advanced Foundations for Knowledge Representation

3.2.1 Patterns and Anti-Patterns

When we create ontology models for information systems, we often assume we know what models we create and how to use them. However upon inspection we may notice that we make many implicit assumptions that are not represented in the model and might create many misunderstandings, communication problems or simply false data. This issue falls into the category of ontology and has to be addressed in order to make accurate models. Prof. Guizzardi³ states the problem definition in his paper as follows: "[...] information systems engineering, in particular, and rational governance, in general, cannot succeed without the support of a particular type of discipline. A discipline devoted to establish well-founded theories, principles, as well as methodological and computational tools for supporting us in the tasks of understanding, elaborating and precisely representing the nature of conceptualizations of reality, as well as in tasks of negotiating and safely establishing the correct relations between different conceptualizations of reality. On one hand, this discipline should help us in producing representations of these conceptualizations that are *ontologically consistent*, i.e., that represent a worldview that aggregates a number of abstractions that are consistent with each other. On the other hand, it should help to make explicit our *ontological commitments*, i.e., to make explicit what exactly is the worldview to which we are committing. In summary, this discipline should help to produce concrete representation artifacts (models) of conceptualizations of reality that achieve the goals of *intra-worldview consistency* and *inter-worldview interoperability*." [Gui14]

This section gives an outline of the work on patterns and anti-patterns to evaluate ontological commitments.

³<http://www.inf.ufes.br/~gguizzardi/Education.htm>

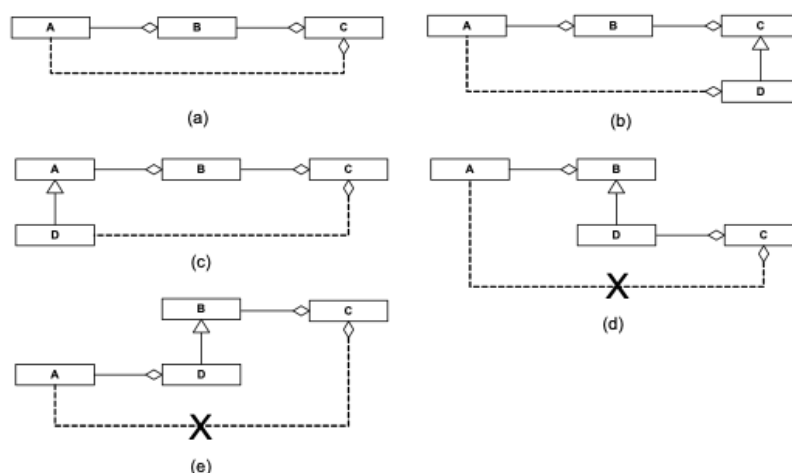


Figure 3.5: Patterns for Identifying the Scope of Transitivity of Part-Whole Relations [Gui14]

Figure 3.5 shows 5 structural patterns. They are context dependent (by giving a structure) and at the same time content-independent (by allowing them to be filled by any class) patterns. As can be seen in figure 3.5.a-c transitivity is allowed and can be deduced out of the given structure, whereas in 3.5.d+e transitivity would lead to false conclusions.

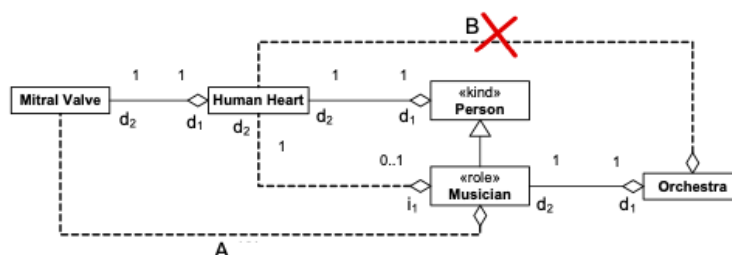


Figure 3.6: Examples of Valid but Unintended Instances of the Organ Transplant Model [Gui14]

Figure 3.6 shows this with an example: Although a human heart is part of a person, and a musician is a subclass of a person (a role to be exact) and a musician is part of an orchestra, it cannot be deduced that a human heart is part of an orchestra (following figure 3.5.d). Part-whole relations have some significance in problem solving tasks and that is why they are frequently addressed in various works. Such patterns can guarantee that certain conclusions aren't made. However it is not possible to prevent that instances of these models will be intended ones. That's where antipatterns are introduced. We can consider another domain with the classes surgeon, transplant, donor and donee, where various relationships can hold between these like hasDonor, hasDonee, hasSurgeon. If

we give those elements a pattern, still many variations can occur. This is exemplified in figure 3.7. In a. you can see that a person can donate a transplant to himself and be the surgeon of the operation of that very same transplantation! In b. the donor is a different person, but the surgeon and the donee is still the same. So as can be seen many 'worlds' can exist for the same structural model. In this case, both are inadmissible following social rules rather than ontological distinctions. With the OntoUML editor⁴ and predefined structural rules and algorithms these 'worlds' can be made explicit and simulated visually. The user can then clear up inconsistencies and correct underconstraining or overconstraining the model.



Figure 3.7: Example (A) and Counterexample (B) of Warranted Inference of Part-Whole Relation [Gui14]

3.2.2 Multi-Level Theory

The OWL standard is continuously revised and enhanced and there are different families of languages. One issue that is not yet addressed in OWL that is being addressed in MLT (Multi-Level Theory) is that various classes can be instances at the same time. Those instances/classes can themselves have instances and classes at multiple stratified meta-levels. This also entails assigning values and properties at those levels.

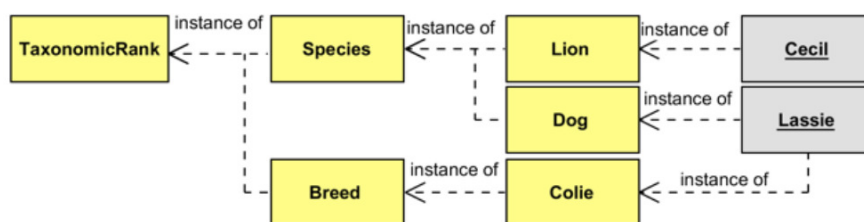


Figure 3.8: A Four-Level Instantiation Chain in a Biological Domain. [Fon+18]

An example is given in figure 3.8. It would not be correct to state that species is a subclass of taxonomic rank or lion is a type of species or cecil is a type of lion. Rather

⁴<https://ontouml.org>

species and lion are instances as well as classes and cecil is just an instance. It would also not be possible to query what species there are if dog and lion would not also be instances. MLT* is the theory that describes rules and structure of the multi-level theory which is then implemented in ML2, the multi-level modeling language (based on XText framework). This well formed theory allows to deal with ordered as well as orderless types simultaneously. These types are different in their in/transitive, ir/reflexive and anti/symmetric properties. Ordered types are schemes to allow for stratification whereas this is absent in orderless types.

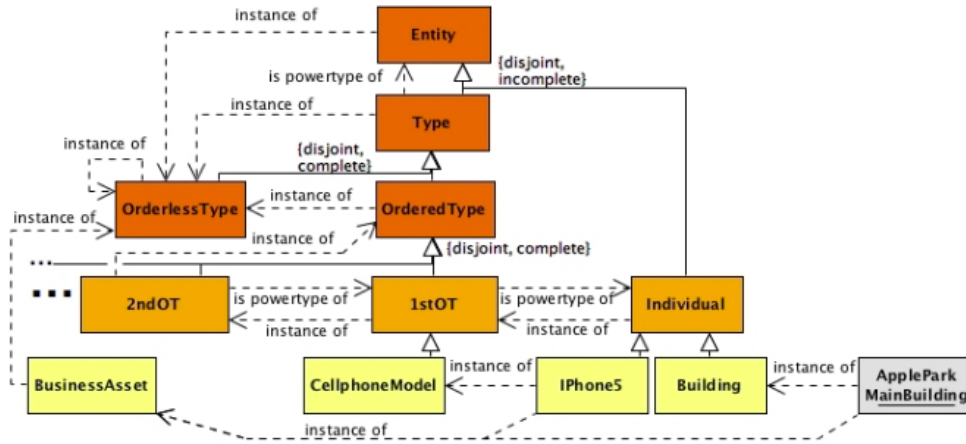


Figure 3.9: MLT* Basic Scheme Extended by a Domain Example [Fon+18]

Figure 3.9 shows this formalized theory with an example. The upper fragments are incorporated in the upper ontology UFO (Unified Foundational Ontology) and the lower fragments are the example. A BusinessAsset of some company for example is an orderless type, because it has instances of various ordered types. AppleParkMainBuilding for example is an individual (an instance of Building) whereas iPhone5 (also an instance of BusinessAsset) is a 1st Order Type (because individual iPhone5's are instances of iPhone5 which in itself is an instance of CellphoneModel). Ordered types are instances-classes at their respective level of stratification. This model is further supported by the powertype pattern [Car88]. Accounting for an upper level abstract syntax and expressive stratifications, MLT* improves on existing approaches.

As discussed in [Joã+18] there are various workarounds for languages that do not support multiple levels that have disadvantages. This paper analyzed wikidata in terms of incorrect conclusion based on insufficient study of MLT. Figure 3.10 shows an example. If Tim Berners Lee is a computer scientist, the subgrouping of the model would suggest that Tim Berners Lee is a profession. Here the error was the subclassing of creator and profession. The multi-level approach would classify profession as second order type and its instances as first-order types. A further paper [Joã+19] discusses the transformation into a common two-level language of the approach.

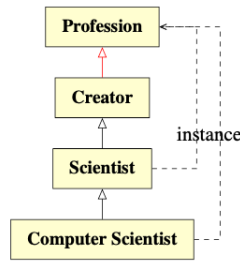


Figure 3.10: The Structure and Occurrences of AP1 in Wikidata (Excerpt) [Joã+18]

3.2.3 Reasoning with Ontologies

Although reasoning is one of the capabilities for introducing and using ontologies, many ontology models do not make use of reasoning. It can be argued that to achieve interoperability, it is not needed. And even in this project usage of reasoning was still to be explored and referenced where seen as appropriate. This section gives an introduction and a few examples of what can be deduced using reasoning. Basically reasoning can be classified into two categories:

1. ontology-based reasoning
2. rule-based reasoning

Whereas rule-based reasoning is addressed elsewhere, here ontology-based reasoning is addressed.

Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human \sqcap Male	$C_1(x) \wedge \dots \wedge C_n(x)$
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor \sqcup Lawyer	$C_1(x) \vee \dots \vee C_n(x)$
complementOf	$\neg C$	\neg Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	{john} \sqcup {mary}	$x = x_1 \vee \dots \vee x = x_n$
allValuesFrom	$\forall P.C$	\forall hasChild.Doctor	$\forall y.P(x, y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	\exists hasChild.Lawyer	$\exists y.P(x, y) \wedge C(y)$
maxCardinality	$\leq nP$	≤ 1 hasChild	$\exists \leq n y.P(x, y)$
minCardinality	$\geq nP$	≥ 2 hasChild	$\exists \geq n y.P(x, y)$

Figure 3.11: Reasoning Constructs Part 1

Figure 3.11 and 3.12 give an overview about basic constructs and come from here.⁵ In figure 3.11 we can see the constructor which is basically how you will find this keyword written in the .owl document. Then there's the description logic syntax, an example, and the first order logic syntax. Intersection, unionOf etc. are known basic constructs in logic. Only if an individual is male and human at the same time (with the intersection)

⁵https://www.kde.cs.uni-kassel.de/wp-content/uploads/conf/iccs05/horrocks_iccs05.pdf

will the SPARQL query find this individual over the superclass (e.g. 'Man'). SomeValuesFrom restrictions exhibit existential dependence and the reasoner would complain if an individual under such a class would not have such a relationship (e.g. does not have a Doctor as child), but only if this is explicitly stated! An allValuesFrom restriction may e.g. either lead to one of those two inferences. With an example: Take a mother, brother and sister, and you have a specific mother, brother, sister etc of each class. If we have 'mother' 'hasDaughter' 'only' 'daughter' relationship, and there is a 'hasDaughter' connection from the mother to the brother, the ontology will infer that the brother is also a daughter. Secondly, only if you make the class 'daughter' and 'brother' disjoint, the ontology reasoner throws an error. This might not be considered by beginners. Cardinality restrictions restrict the number or relationships to a given class. It has to be considered that all of those individuals have to be made explicitly different in order that the restriction can serve its purpose.

OWL Syntax	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human \sqsubseteq Animal \sqcap Biped
equivalentClass	$C_1 \equiv C_2$	Man \equiv Human \sqcap Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter \sqsubseteq hasChild
equivalentProperty	$P_1 \equiv P_2$	cost \equiv price
transitiveProperty	$P^+ \sqsubseteq P$	ancestor ⁺ \sqsubseteq ancestor

Figure 3.12: Reasoning Constructs Part 2

Figure 3.12 shows further constructs in OWL syntax (same as above constructor). In OWL, a rich subClassOf hierarchy (which can be nested) yields the meaning and understanding for the individuals. It is an essential part of the reasoning process. They describe as subsumptions which concept is more general than the other. SubpropertyOf works similar to "subclassOf" but with object properties. An equivalent class or defined class, allows to infer class hierarchies and if individuals are implicitly part of a class. E.g. an equivalent class 'mother' 'hasDaughter' 'some' 'daughter' makes as subclass all classes that have an 'hasDaughter' 'some' 'daughter' restriction. Furthermore, individuals that have a daughter will be inferred to be mothers. Related to that is the equivalentProperty which makes two classes equal. In addition, transitive properties allow for chain inference in a bottom-up or top-down manner. E.g., that when an engine is part of a powertrain and a powertrain is part of a car, it is inferred that that engine is also part of the car. Which is not stated explicitly! Additionally, there is also the inverse property e.g. something which has a part is in turn the part of something.

3.2.4 Getting an Overview of Foundational Ontologies

Foundational (or upper or top-level) ontologies are ontologies that consist of very general terms, such as object, property or relation. They are used to help making ontologies interoperable. They are a rather recent development in ontology engineering. Some academic

sources seem to be discordant about their usefulness on the one hand. But some leading professors in ontology engineering are convinced about their usefulness. The trend seems to favor using top-level ontologies. One recent assessment was made by Maria C. Keet in [Kee11], [Kee18]: "A controlled experiment has been carried out with 52 novice ontology developers, which showed that, on average, using foundational ontology resulted in an ontology with more new classes and class axioms, and significantly less new ad hoc object properties than those who did not, there was no part-of vs. is-a mistakes, and, overall 'the cost' incurred spending time getting acquainted with a foundational ontology compared to starting from scratch was more than made up for in size, understandability, and interoperability already within the limited time frame of the experiment." Some of their advantages are listed in above section, which lead to a decision to utilize top-level ontologies in the project. This author thinks it is a wise and farsighted decision to make use of them. Some of the better-known top-level ontologies are:

- UFO (Unified Foundational Ontology) [Gui05]
- BFO (Basic Formal Ontology) [ASS15]
- GFO (General Formal Ontology) [HH06]
- DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering)
- BORO (Business Objects Reference Ontology)
- SUMO (Suggested Upper Merged Ontology)
- YAMATO (Yet Another More Advanced Top-level Ontology)
- GIST (Minimalist upper ontology) [McC10]
- GUM (Generalized Upper Model) 2.0 [BHR95]

This is not a complete list, but a comprehensive list would also not list more than around 20 top level ontologies. Although being domain neutral top-level ontologies suffer themselves from the problem that they are accounting for different nuances and are themselves not interoperable. One study showed that it was possible to align 30% of two top-level ontologies, but most parts were not compatible. There are efforts to unify these top-level ontologies, but there does not seem to be a consensus if this is even possible. At this time, efforts are being made to unify a core of these ontologies.

3.2.5 BFO Structure and Explanations

BFO is a so called realist ontology and contains a minimum of upper level classes. Part of it are also 'definition' and 'example of usage' properties that help in building the ontology. Figure 3.13 gives an overview of its classes. Displayed here is the **continuant** part: entities that continue/persist to exist through time completely. The **occurent** part is not contained in the figure: it describes entities that occur/happen in time like events and processes. **Independent continuant** is a continuant that is the bearer of some role, quality, disposition etc. and exists independently of those qualities. The dependent entity is secondary in dependence. **Generically dependent continuant** is a continuant that is dependent on more than one independent continuants (that can serve as its bearer, or more precisely that are able to migrate from more than one independent continuants).

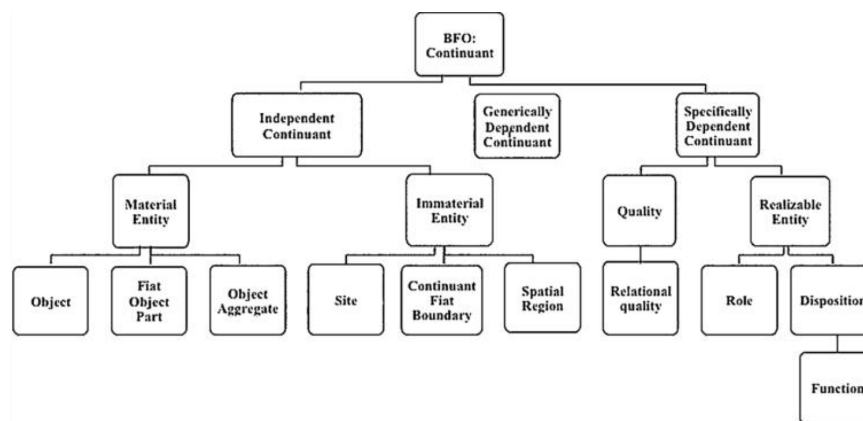


Figure 3.13: Basic Formal Ontology Continuant Part [ASS15]

Example is a pdf file on serveral storage discs. **Specifically dependent continuant** is a continuant entity that depends on one independent continuant for its existence. They exhibit existential dependence, e.g color of this tomato, the pain in your left elbow, mass of this cloud. **Material entity** is an independent continuant that has some portion of matter as parts. An **object** is spatially extended in 3 dimensions, causally unified and maximally self connected. E.g. a person with a hat is not an object, but a person and a hat are objects. **Fiat object part** is a material entity that is a proper part of some object, but is not demarcated from the remainder of this object. E.g. upper torso or western hemisphere. **Object aggregate** is a material entity that is a collection of objects. E.g. a company or organisation. An **immaterial entity** contains no material entities as parts. A **site** is an immaterial entity in which objects can be contained, defined relatively to some material entity like a wall or floor, which builds the containment, e.g. a cave. **Continuant fiat boundary** is an immaterial entity that is of zero, one or two dimensions and does not include a spatial region as part. E.g. objects boundary (surface) where it meets its surroundings. **Spatial region** is an immaterial entity which is part of space. **Quality** is e.g. color of this portion of blood, shape of this hand, mass of this kidney. **Relational quality** have a plurality of independent continuants as their bearers. E.g. marriage bond, instance of love, being a parent of. A **realizable Entity** is a specifically dependent continuant that inheres in one or more independent continuants (as qualities), in contrast to qualities, realizable entities are exhibited only through certain characteristic processes of realization (manifested, actualized, executed), e.g. role of being a doctor, the functions of the reproductive organs, the disposition of a portion of blood to coagulate, the disposition of a portion of metal to conduct electricity. A **role** is an externally grounded realizable entity, which is possessed by its bearer because of some external circumstance. A role is assigned by some other person to execute some form of authority, e.g. role of being a doctor. A **disposition** is a realizable entity in virtue of which a process of a certain kind occurs. The trigger might consist of being placed in a certain environment, internal event within the object or external

influence, e.g. the disposition of a magnet to attract iron fillings, A **function** is a special kind of disposition, a realizable entity whose realization is an end directed activity. It is something the bearer possesses because of how it came into existence (naturally or through design). Or came into being in order to perform activities of a certain sort, e.g. a function of a hammer to drive in nails, a function of a pen to write.

These upper level structures contain disjoint hierarchies and thus support basic reasoning processes when subclassing other classes under these top classes. E.g. the ontology does not allow for instantiating both a hardware part and a function as superclass of the same instance, because independent continuant and specifically dependent continuant are disjoint.

3.2.6 Part-of Relationship considered more closely

In the GENIAL! basic vocabulary we defined the basic transitive relationship has-part with its inverse part-of and the non-transitive relationship part-of-directly. Part-of-directly is used here to define the relationship between two classes that is a direct (one level hierarchy) relationship between components. That was necessary in order to conform to the requirements of description logic.

This was expected to suffice for the use cases we face in the project. However, the has-part relationship is much richer and between different kinds of classes different relationships hold that may not easily be combined. Here some examples are given.

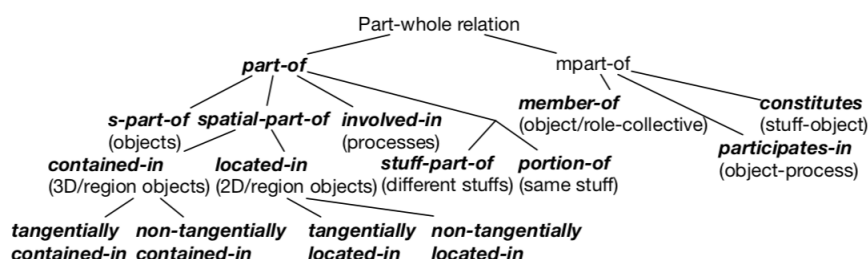


Figure 3.14: Part-Whole Relationships Taxonomy [Kee17]

In [KA08a] Maria C. Keet describes ontological distinctions of the part-of relationship. Some part-of relationships (if the ontology just contains part-of) may simply be false. Also, she makes much use of domain and range axioms to specify the use of each ontological distinction. Figure 3.14 shows a taxonomy of part-whole distinctions. These distinctions and the domain and range axioms are based on the foundational ontology DOLCE. The involved-in object property holds between perdurants (both as range and domain). Perdurants are similar to the occurents described in BFO. As an example 'chewing' is involved-in (part-of) the greater process of eating (which as processes are both occurents/perdurants). In this way, 'chewing' for example could not be involved in a physical object. Keet describes these relationships in detail with their respective formalizations.

3.2.7 Other Developments

Distributed Ontology, Model, and Specification Language (DOL)

DOL is a metalanguage that is unified in order to allow to integrate and reason over a variety of languages of logic. This reasoning may not always be successful, but DOL allows to express some axioms that are not part of OWL in another language and then link them together. The semantic web community has different requirements/statement of affairs on language that go beyond OWL, but can be expressed through a combination of logics.

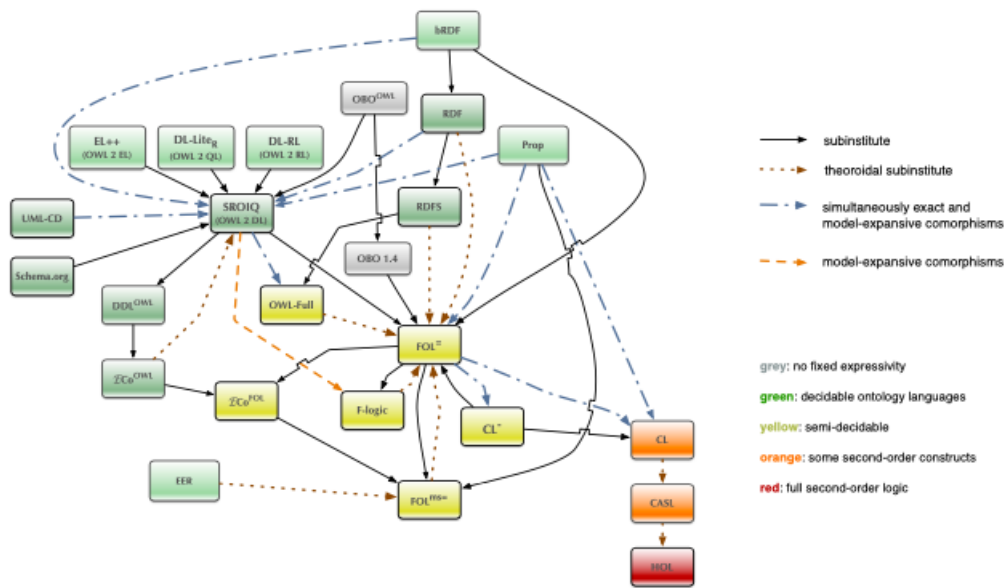


Figure 3.15: Family of Languages/Logic supported by DOL with its Translations (Arrows) [Mos+14]

Fuzzy and Rough Ontologies

Fuzzy and rough ontologies deal with uncertainty and vagueness. When something is uncertain, we lack knowledge about it and may only be able to ascribe a possibility/probability to a true-false statement. For example, a *low* blood pressure, a *green-ish* car or an event that occurs *often*. When something is vague, the concept in itself is not well defined, may be categorised differently by different people, and may yield different results in different contexts. Vague terms are for example 'young', 'cold' or 'tall'. The reason this is important becomes obvious if we for example want to retrieve a query like: 'What *cheap* hotels are *close* to the train station?'. This also fits in nicely with the notion of context. If you just bought a car or are broke, or are at the location by

foot instead of by bike, these concepts may take on a different meaning and shift to its opposite (e.g. something is 'expensive' rather than 'cheap' or 'far' instead of 'close'). The fuzzy concepts of fuzzy ontologies have an interpretation function that varies between $[0,1]$ that is usually a continuous function indicating the degree to which a concept is true or satisfied. There are reasoners like fuzzyDL reasoner who can reason with this information, which is often annotated. Rough ontologies account for that we may not know if something is an instance of a class. The properties of this class can then be specified more completely to increase precision.

Time and Ontology

How to model aspects of time, how to reason with it and temporal description logic are subject-matter of research. The time ontology⁶ is a way to describe time in a way that it can be utilized consistently across different domains and applications. It describes classes like Interval, TemporalEntity or Instant and object properties like before, after or duration. It also supports basic reasoning tasks. An issue that multi-level theory solves, that is present in this ontology: You may have a class DayOfWeek and an instance Friday. But you are not able to instantiate specific Fridays, e.g. Friday the 6th. This is because Friday is not a first-order type. To exemplify why the encoding of temporal information is important and what they can enable is shown with the two queries [Kee18]: 'Who was the South African president after Nelson Mandela?' and 'Which library books have not been borrowed in the past five years?'

SHACL - SHapes Constraint Language

Whereas OWL is designed for inference, SHACL⁷ is based on RDF and is for validation purposes. To use validation, one has to move from an open world to a closed world assumption. The difference can be most easily explained by: in OWL if something has certain properties it belongs to a class, whereas in SHACL it is if something belongs to a class it has to have certain properties. It is kind of the opposite logic. SHACL can tell you if a property is missing or has a wrong value or has a wrong format. The basic construct is a shape which can be either classes, nodes, the object of a property or the subject of a property. When the shape is the object of a property, we can for example define that as soon as someones 'works for' something, he has to have a name, e.g. a string. It uses Paths to define the route of constraints which can be predicates sequences or inversed paths. There are filters who further constrain a shape to narrow it. SHACL allows multiple people to use their own views as 'shapes' and thus use the same ontology to satisfy their needs: one ontology - multiple shapes. In companies different people have different understanding and meaning for different terms. Shapes allow that such understandings can be met and work together. So that these different people can validate according to what is important to them. These shapes can be reused and deactivated.

⁶<https://www.w3.org/TR/owl-time/>

⁷<https://www.w3.org/TR/shacl/>

Chapter 4

Tools and Methods for Knowledge Management with AgilA

4.1 AgilA Introduction

The tool is best introduced from an excerpt of the introduction of the first paper of the author about the tool [WG18]: *"Digitalization, especially under the umbrella of 'Industry 4.0' has become very popular and quite a buzzword. In particular this regards to production, logistics and maintenance. The deep networking and application of intelligent methods yields opportunities to reduce costs and shape more flexible productions. The life cycle of complex products like in automotive industries, especially comprises product development. For product development the focus of research has been model-based development; virtual prototypes allow for rapid evaluation of designs and corporate and shared development of hardware, software and mechanical parts.*

A big issue of model-based development is that the design and development of such models is time consuming and costly, because it is aimed for modeling the dynamic behavior as meaningful as possible. This often requires knowledge of the whole context of a system and is non-trivial. Often models are missing completely, for example from mechanical parts and their tolerances. Here, networking of development with production, product life cycle management, logistics and other stakeholders yield new opportunities.

In the scope of this publication a methodology and a prototype knowledge-based tool is introduced, that depicts ways to address aforementioned problems through missing models. The tool will be demonstrated according to an example from the automotive domain. By means of the example, it will be demonstrated how models with dynamic behavior in development processes are substituted through properties, uncertainties and a constraint net of a knowledge-based system.

AgilA (Agil Automotive) is based on a knowledge base in form of ontologies and saved in the Ontology Web Language (OWL). The knowledge base consists of the knowledge about the structure of considered parts. Furthermore, it consists of calculation functions, which describe a relationship between properties of parts of the system and system properties and functions. Calculation functions are often known to engineers as approximate

estimation functions. Additionally, there are more uncertainties and tolerances quantifiable especially from production and product life cycle management. The tool gathers this knowledge in a web-based database and allows for the rough, but holistic optimization of a system.

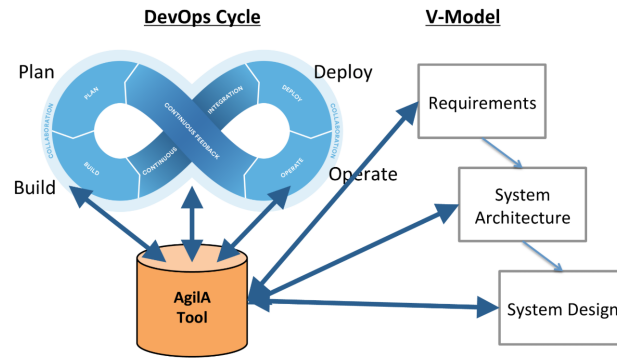


Figure 4.1: Scope of Tool AgilA [WG18]

Figure 4.1 gives an overall overview of the scope of the tool. As can be seen its application is to be positioned in early stages of the development process. It is used to define and refine the requirements and make early estimations of the system architecture and the design. By using just semantics to specify the system it is supposed to be used in the development and operation cycle as well, where we directly deploy new attributes and definitions and immediately run them on our system. The motivation and usefulness of the approach have been validated and tested and have been elaborated extensively elsewhere [GW18].

4.2 Preparatory Work

To kickstart the work, we began implementing two versions of a graphical user interface that allows for some basic setting up and configuration of an automobile with its features and variables. The first was implemented by the author and the second one by students. The author's version as seen in figure 4.3 relies on selecting systems and system parts from listview panes which get instantiated as individuals in the ontology by selecting them (e.g. in the picture a BMW as designed systems is selected as well as a chassis configuration). In the lower part of the picture, it can be seen that the chassis is refined to a sport chassis and the system speed is calculated depending on the components.

In the graphical user interface of the students in figure 4.2, it can be seen that all properties from the ontology for every individual on the right side is displayed and that they can be tweaked and edited in the other 'add to ontology' tab. On the left the class hierarchy is displayed with all classes in the ontology. On the bottom we can see that

object properties can be selected and added to the properties of the cars. Both implementations use a different ontology and have a slightly different setup.

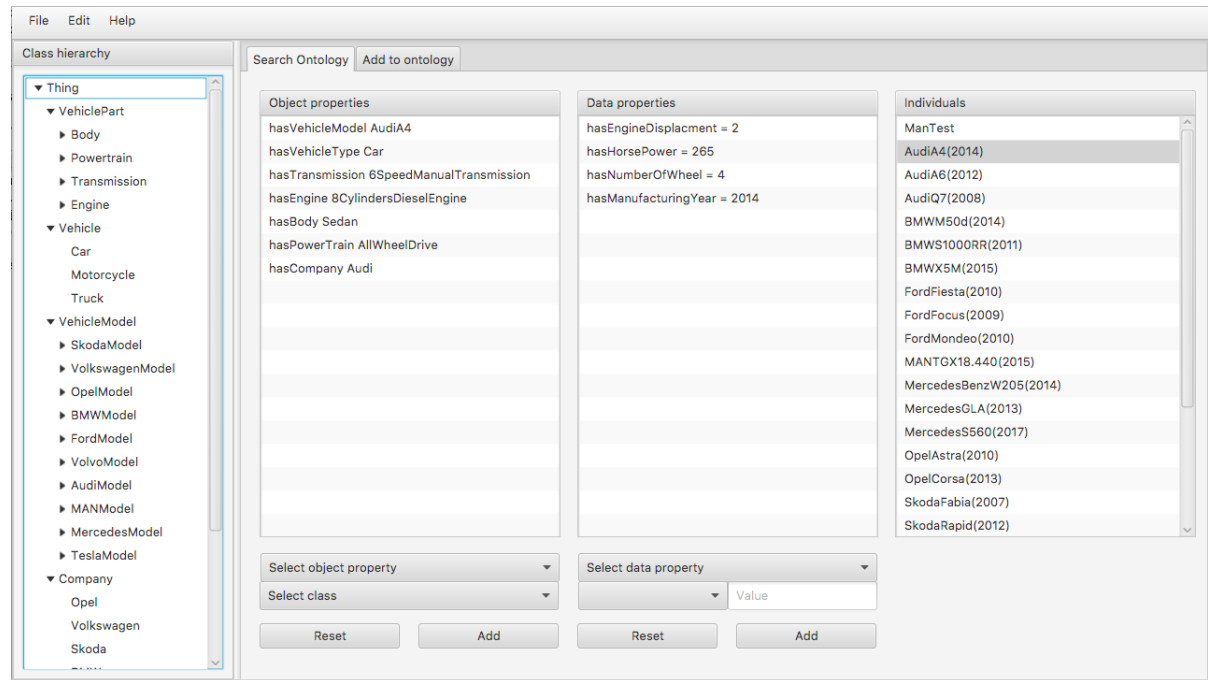


Figure 4.2: Graphical User Interface of Students [WG18]

4.2.1 Initial Architecture

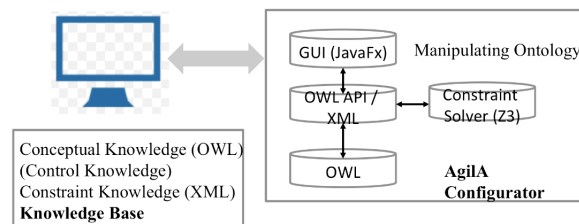


Figure 4.4: Initial Architecture of AgilA

The initial architecture just consists of a local AgilA Configurator, a software program running on a desktop machine. Which in itself consists of a graphical user interface as illustrated previously, the OWL API to control the ontology saved in the Ontology Web Language (OWL) and the conventional constraint solver Z3. The conceptual knowledge of the system and its parts were stored in OWL, whereas the constraints, their formulas

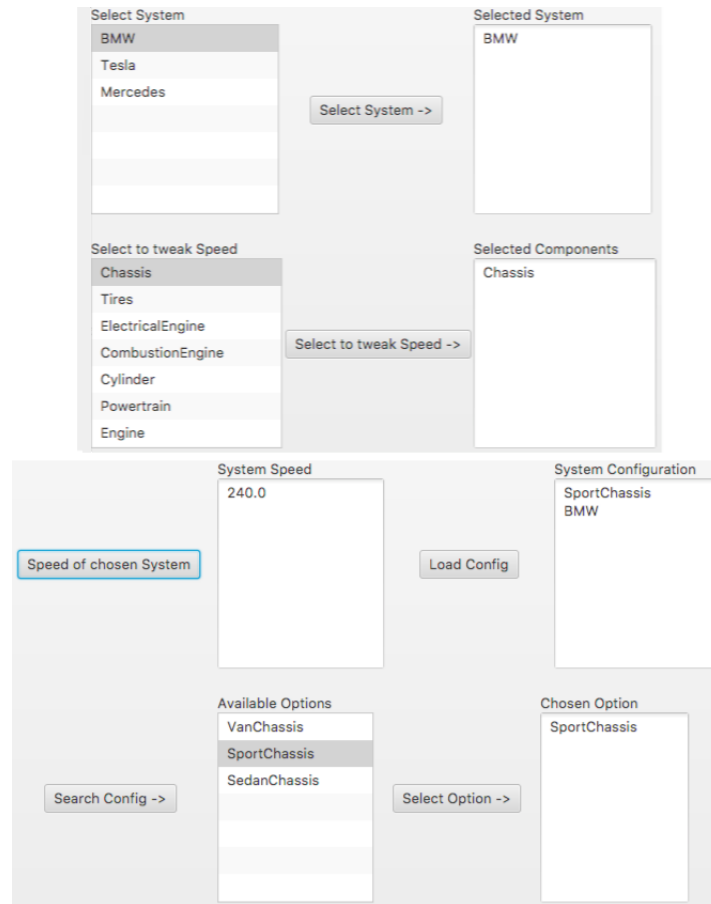


Figure 4.3: Graphical User Interface of Author

and semantics were stored initially in XML. This was to be able to parse the formulas and translate them to Z3 in a machine-readable fashion. The control knowledge is in brackets because it is required by the expert system, but was only partially implemented in the initial version. The architecture is illustrated in figure 4.4

4.2.2 Ontology

From the paper "A Knowledge-based Approach for Engineering 4.0" from the author [WG18]: "The structure of cars as well as their components are constituted by *is-A* and *hasParts* relationships and allows for a specialization ("Top-Down") as well as integration (Bottom-Up) of the system and their components.

Figure 4.5 shows an excerpt from the conceptional structural hierarchy of the engine. Here within the graphical overview the relations "*is-A*" (yellow) and "*hasParts*" (orange) are illustrated as edges and the classes as nodes of a graph. Within the graphical view other relationships and classes are hidden that are in relationship with these classes. This

is indicated by the plus symbol on the corresponding class. This hides the complexity and allows for just showing classes and relationships of interest.

Figure 4.6 shows an additional overview of how the class `car` is structured with its components `chassis`, `powertrain` and `body` and again how a `powertrain` consists of `tires`, `gear`, `coupler` and `engine`. The `engine` in itself can again be specialized into an `electrical` or `combustion` engine (see Figure 4.5). The concepts `System` as well as `SystemPart` are utilized by the `userinterface` to display the system and components in lists. The class `engine` can be further stripped down even to control units."

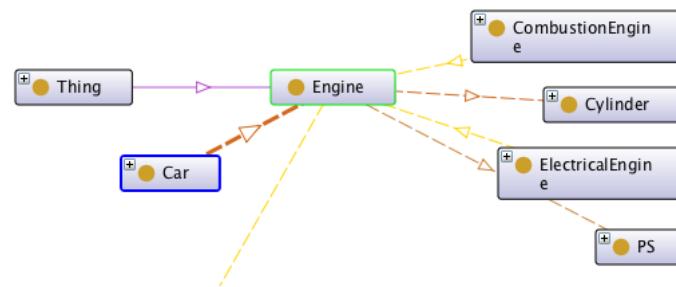


Figure 4.5: Structural Hierarchy of the Engine via is-A and hasParts Relationships [WG18]

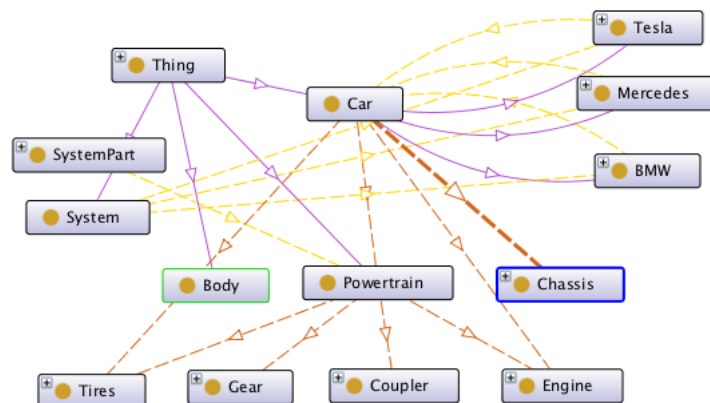


Figure 4.6: Overview of the Components of the Ontology [WG18]

4.2.3 Constraint Net

To demonstrate and test the methodology, at least in the way to show its functionality, a hypothetical constraint net was set up. From the paper "A Knowledge-based Approach for Engineering 4.0" by the author [WG18]: "*To configure a car there are a variety of dependencies, constraints and features. A first, preliminary overview gives Table 4.1.*

Table 4.1: Constraint Net of Use Case [WG18]

Pin	Value	C0	C1	C2	C3	C4	C5	C6
Car Price	[15.000-70.000 €]	x						
Speed	[190-240 km/h]		x	x				x
Power	[150-220 PS]		x			x		x
Fuel Usage	[3-8 l/100km]			x			x	
Engine Price	[500-1000 €]	x				x		
Car Body Price	[1000-2000 €]	x						
Tires Price	[50-200 €]	x						
Car Type	[Type A]				x			
Engine Type	[Type A]				x			
Mass	[1.200 - 2.200 kg]						x	x
Driving Resistance	[0-2500 N]							x
Frontal Area	[1-4 m ²]		x					x

Furthermore in the following there will be an explanation of all dependencies via formulas, which are listed in the table with Constraint (C) 0-6. The constraints get then translated into the knowledge base and are solved with the external constraint solver Z3 and with Affine Arithmetic, which is both work in progress.

$$Car.Price = Engine.Price + Body.Price + Tires.Price \quad (4.1)$$

Constraint C0: All round price. For reasons of simplicity only three components are considered.

$$v = \sqrt[3]{\frac{2P}{\rho AC_w}} \quad (4.2)$$

Constraint C1: Dependency of speed and power. v: speed, P: power, A: frontal area of the car, C_w: air drag coefficient, rho density of medium air.

$$y = a(x - 40)^2 + b \quad (4.3)$$

Constraint C2: Dependency of speed and fuel usage. y: fuel usage, b: fuel usage at 40 km/h and according to type of car, a: factor depending on gear, x: speed.

Car type	Fuel consumption in liter/100km at 90 km/h	at 120 km/h
Tesla	-	-
BMW	5,7	7,5
Mercedes	11,6	12,5

Table 4.2: Fuel Consumption of Cars [WG18]

Table 4.2 illustrates the fuel consumption in liter/100km according to the speed. This is a sample which allows to calculate the variables a and b; for example the result for the BMW is a=0,0005 and b=4,5463.

Constraint C3: is a simple dependency of types. There are certain engine types that suit certain cars.

$$P = ax \quad (4.4)$$

Constraint C4: Dependency between power and price. P: power, a: proportional constant, x: price. Here the result was linearized for simplicity, but it can also be taken from a relational table.

$$y_{add} = m_{car} + m_{add}k \quad (4.5)$$

Constraint C5: Dependency of mass and fuel usage. yadd: additional fuel consumption, mcar: mass of car, madd: additional mass in 100kg, k: proportional factor, i.e. 0.5 liters in 100km.

$$P = C_r mgv + \frac{A}{2} C_w D v^3 \quad (4.6)$$

Constraint C6: Dependency of power in relationship with speed and driving resistance. P: power, m: mass, v: speed, g: weight force, Cr: rolling resistance coefficient, A: frontal area, Cw air drag coefficient, D: density of medium air ($1,29kg/m^3$).

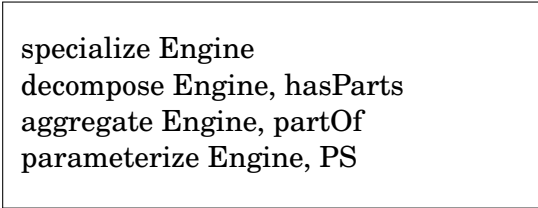
4.2.4 Expert System Implementation

In this section we see a further development of the graphical user interface from a pure manipulation and tweaking of the ontology to the construction of a technical system with concrete construction steps. From the paper "A Knowledge-based Approach for Engineering 4.0" by the author [WG18]: *"This section describes the construction process inherent in AgilA. It is quite similar to that of known expert systems. The construction is realized by instantiating concepts, which yield a partial or complete solution depending on the conceptual hierarchy. The process is similar to the one in [GCS90] and has been adapted to our use case and constraint net and is illustrated with some basic examples. It is to be noted that it is possible to engage interactively on all the following criteria by the user via the user-interface and set up new attributes, selection criteria, phases etc."*

Agenda

Here the current (partial) solution is being analyzed in regard to which construction steps to take. The agenda lists all executable construction steps possible to take in the current situation. In the case of our use case a possible agenda looks as follows tailored to the engine:

As can be seen from the example, the agenda offers ways to specialize an object, decompose an object via a top-down approach, aggregate an object via a bottom-up approach and parameterize objects, especially their attributes like for example here the horsepower of the engine (here german for PS) which is either defined or restricted.



```

specialize Engine
decompose Engine, hasParts
aggregate Engine, partOf
parameterize Engine, PS

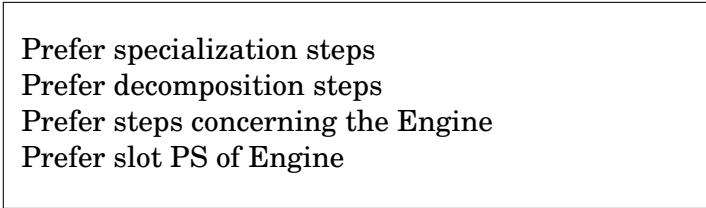
```

Figure 4.7: Agenda for current Engine Instance [WG18]

Here the conceptual hierarchy describes the set of all admissible constructions, as to help specify and define the agenda.

Selecting from the Agenda

Whereas the agenda lists all possible construction steps, the next step is to select one of these steps from the agenda by agenda selection criteria as seen in Figure 4.8. These preferences may depend on the collaborator using the tool and the actual stage he is working on. Steps can be selected by pattern or rating, and what might also play a role in this step are elements from recommender systems. When there are equally rated steps further criteria are used to determine which step to prefer first.



```

Prefer specialization steps
Prefer decomposition steps
Prefer steps concerning the Engine
Prefer slot PS of Engine

```

Figure 4.8: Some Agenda Selection Criteria, also concerning the Engine [WG18]

Value Determination Methods

Value determination methods are important when it is not possible to make a fixed decision on a construction process, and there are several conflicting options. To help with decision making in this process a value determination method is used, which can be one of the following things: interactively asking the user to make a decision, propagate the decision from the constraint net, evaluating a function, using a default value, querying an external source, or others.

Conflict Resolution Knowledge

In the current implementation conflict resolution knowledge is determined by the constraint solver, when a value or its propagation is out of bounce. In this step it can be determined which steps to take to resolve that conflict and its cause is analyzed. It is currently implemented as rules and shown in Figure 4.9 as a possible conflict of the fuel usage attribute. It is possible to determine domain dependent rules for specific purposes.

Phases and Strategies

It is useful to pack up control rules in phases, because each phase has special steps to

<p>If Fuel Usage of Engine too high Cause 1: Speed Attribute Cause 2: Power Attribute Cause 3: Mass Attribute Proposed Solution: reduce value</p>

Figure 4.9: Example of Conflict Resolution Rule [WG18]

take and a construction process consists of several phases. In this step we sum up various criteria mentioned above like selection criteria, the focus of the construction, how to determine certain values (e.g. by asking the user or through constraint propagation), the construction context and conflict rules.

Control Rules

To switch between phases with smart knowledge, control knowledge is needed. In this way it is possible to select and terminate a phase. These rules are listed in control knowledge and an example is given in Figure 4.10."

<p>If all components of Car are specified And Powertrain is incompletely specified Do Strategy Powertrain Phase</p>

Figure 4.10: Example of Control Rule [WG18]

4.3 Distributed GUI

4.3.1 Refined Architecture

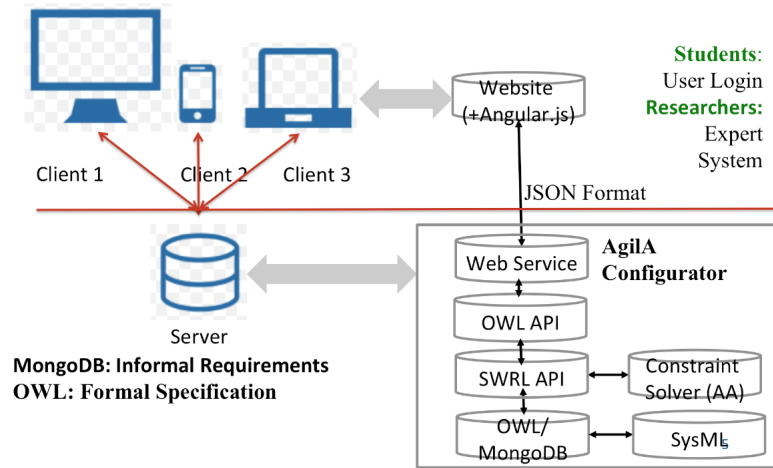


Figure 4.11: Refined Distributed Architecture [WG18]

Figure 4.11 shows the refined architecture, which caters to the need of a collaborative workflow between peers. It is a distributed architecture that consists of a client side, which can be accessed through various devices and a server side which runs the program. On the client side we have a website, which displays the constructs of the ontology and can make adjustments to the system similar the one introduced in the initial architecture. It gets its data send via JSON format from the server. On the server runs the AgilA configurator which is now refined from the local GUI. It runs a web service, which communicates with the client and hands over the data from the ontology which he gets via the OWL API. Additionally, now the SWRL API reads the constraints from the ontology, which are stored as rules and forwards them to the web service. Besides the formal specification of the system, we store informal data and requirements in a NoSQL database as well and translate SysML models to OWL¹. The user login was specifically designed from students and the expert system was done by researchers.

4.3.2 Modeling Constraints as Rules

It was considered how to best implement the constraints. One of the arguments was, that it would be practical to have a common data format for all the knowledge and not just have data distributed, partly in XML. Another argument was that the semantic meaning gets lost in XML, and it would thus be more practical to have it in the ontology as well. One way of achieving that was by utilizing the semantic web rule language and to employ the constraints as rules in the ontology. OWL lacks the expressiveness to store

¹<https://github.com/brunopessanha/OWLRevelio>

Data property assertions

- hasDragCoefficient 0.26f
- hasHorsePower 310
- hasFrontArea 2.61f
- hasMaximumVelocity 81.30755f
- hasPowerInWattsDecimal 228004.61250

MyCar

- 'AudiA4(2014)'
- 'AudiA6(2012)'
- 'AudiQ7(2008)'
- 'BMW50d(2014)'
- 'BWS1000RR(2011)'
- 'BMW5M(2015)'
- 'FordFiesta(2010)'
- 'FordFocus(2009)'

Constraint 1: ConvertPowerAllIndividuals

Name: ConvertPowerAllIndividuals

Comment:

Status:

Ok

owl:Thing(?t) ^ autogen0:hasHorsePower(?t, ?horsepower) ^ swrlb:multiply(?powerinwattsdecimal, 735.49875, ?horsepower) -> autogen0:hasPowerInWattsDecimal(?t, ?powerinwattsdecimal)

$$P = ax$$

Constraint 2: MaximumVelocityForAllIndividuals

Name: MaximumVelocityForAllIndividuals

Comment:

Status:

Ok

owl:Thing(?t) ^ autogen0:hasDragCoefficient(?t, ?dragcoefficient) ^ autogen0:hasPowerInWattsDecimal(?t, ?powerinwatts) ^ autogen0:hasFrontArea(?t, ?frontarea) ^ swrlb:multiply(?mul1, ?powerinwatts, 2) ^ swrlb:multiply(?mul2, ?frontarea, ?dragcoefficient, 1.25) ^ swrlb:divide(?div1, ?mul1, ?mul2) ^ swrlb:pow(?maximumvelocity, ?div1, 0.33333) -> autogen0:hasMaximumVelocity(?t, ?maximumvelocity)

$$v = \sqrt[3]{\frac{2P}{\rho AC_w}}$$

Figure 4.12: SWRL Constraints

mathematical constraints, which is yielded by SWRL. In the ontology itself it would just be possible to express that a certain variable is part of a formula and that formulas can be queried according to a variable like comprehensive approaches to establish a semantic mathematical library like the one in [Nev+13].

It was chosen to model constraints in the ontology as two separate datatypes with lower and upper bound respectively. Figure 4.12 shows two example constraints modeled in the semantic web rule language. The first is a power constraint to calculate the power of a car in different units and secondly a maximum velocity constraint which is an output of the power of the car, the drag coefficient, the frontal area and the density of the medium air. In the upper half of the illustration on the right side there are the individuals with the individual 'MyCar' to which the constraints apply and on the left side are the data type properties with the concrete values of the variables. With yellow background are the data types which are inferred from the reasoner with the applied rules. The constraints are directly calculated by the ontology/reasoner. This might be redundant since the constraint solver does the same thing, yet it cannot calculate uncertain boundaries when the variables are not all specified. However the constraint knowledge is still present and can be read. The other task is reserved for the constraint solver. The SWRL rules applied here use the mathematical built-ins swrlb:multiply, swrlb:divide and swrlb:pow which are self-explanatory and similar to the math functions in java.

Chapter 5

Knowledge Representation and Models in GENIAL!

Most content in this chapter served as a journal submission and is contained there in an identical, corrected, similar or enhanced version [[Waw+](#)].

5.1 Roadmapping Introduction

Automotive development, planning and innovation has many influencing factors and requires increasing collaboration along the value chain. Shortened development cycles and disruptive technologies yield difficult to predict circumstances. One essential factor that is driving the automotive industry in Germany is to understand what shapes micro-electronic development within the next 20 years. To tackle this roadmaps are created by leading experts in the field that define the goals and strategies necessary to secure the economic leadership of the industrial sector in the country. In the case of micro-electronics disparate factors influence developments. Roadmaps have so far thus been informal documents, which are updated in periods of several years to live up to most major changes. The project GENIAL! contributes here by providing a digital, 'living' roadmap which is integrated in the PLM (product lifecycle management) and estimates alternative implementations by considering context and dependencies in the form of constraints, among other criteria. This entails estimating which technology is suitable in certain circumstances, as well as exchanging and collaboratively refining innovations and their implementations. This resulting tool is designed to help developers and managers when dependencies are too complex to overlook. Thus one part of the solution is sought in calculating mathematical dependencies via constraint solvers that enable one to solve and optimise constraint nets via specific ranges and properties. Set languages that support reaching the overall goal are the system modeling language (SysML), the ontology web language (OWL), and eventually newly developed languages. Here, the focus on the role and use of ontologies in the project. Fig 5.1 depicts how the tool is used by end users, developers, designers and experts. The knowledge base contains architectures, variants, simulation models, rules, properties and functions, the output consists of feasi-

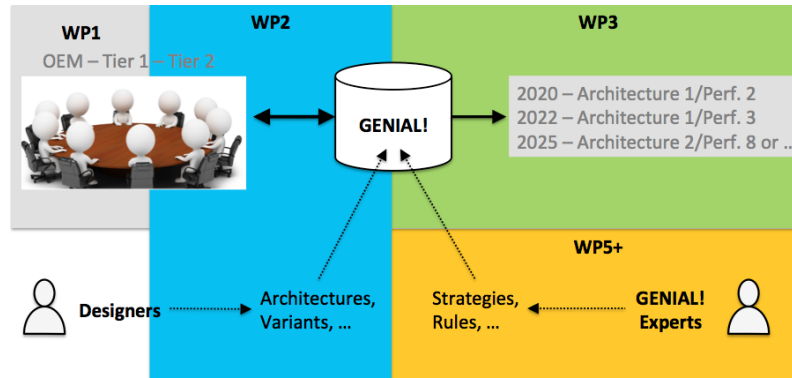


Figure 5.1: Different Parties using GENIAL! Tool

ble architectures, solution spaces, properties and performances. The figure further shows the partitioning into workpackages (WP). GENIAL! stands for Common Electronics Roadmap for Innovations in the Automotive Value Chain. It considers changing value chain structures as well as cartel and competition law. [GWZ19] and [GW18] further explains basic ideas, concept and preliminary implementations.

5.1.1 Role of Ontologies in GENIAL!

The basic principle for our decision to utilize ontologies in GENIAL! was that they transcend the purpose of a single project. They should be developed and maintained independently, merged with other knowledge, and yield additional value via their modularity and extendability. Their integration to PLM, DevOps (Development Operation Cycle), expert systems with their rules, and self-aware cyber-physical systems and interoperability in general. They are able to define the meaning of the data, can be interacted with in natural language and serve as a single source of truth and unified data model. They are able to capture (most) pieces of information and make the computer understand and process it. It was desirable to have a library of components that denoted ideally the same thing and standard used across companies, domains and disciplines. Initially there were discussions on whether to use OWL or SysML or it focused on what way they could complement each other. The authors found that there is still confusion about the purpose and functionality of both languages to engineers. Table 5.1 gives a compiled comparison that can serve as a guideline. Other works that did a more comprehensive analysis are [Gra09], [ZL12] and [KA08b]. Various diagram types have been translated between both languages ([Gra09], [Isa15], [BBM14]) and there are similarities. E.g. the information of block diagrams and internal block diagrams can be translated sufficiently similar.

SysML	OWL
Machine readable / executable	Machine understandable
No reasoning capabilities	Reasoning and inference
Focus on modeling	Focus on semantics and knowledge
Graphical	Non-graphical
Closed world assumption	Open world assumption
Not queryable	Dedicated query language and API
System, its components, connections and requirements	Things and their relationship
Dedicated to represent social and technical systems	Dedicated to represent all information
Good for documentation / interfaces / simulation	Good to represent context and serve as standard, interoperability
More concrete less considerate	More abstract and ontologically correct
Collaboratively work on	Share, exchange and extend
Used by humans	Used by autonomous agents, devices and computers
Designed by humans	Designed by humans
More informal	More formal
Origin in systems engineering	Origin in artificial intelligence

Table 5.1: Comparison between the System Modeling Language and the Ontology Web Language

5.1.2 Key Factors for Shaping Microelectronic and Automotive Development

The most significant part of a digital roadmap is the key contributing factors that shape microelectronic and automotive development. As a first step, the ZVEI technology-roadmap¹ 'Next Generation - electrical components and systems' (translated from german) is utilised as input. Relevant future input is further expected from the "Arbeitskreis Automotive"², which serves as the Advisory Board of the project. The ZVEI study identified trends and mega trends, which are trends who have a great and epoch making character, can be observed over a period of decades, and can create fundamental and deep change. They transform whole societies. Trends (with mega trends in bold) identified are (1) **globalisation**, (2) mobility, (3) **demographic development** (including (4) aging of society, (5) growth of world population), (6) health, (7) urbanisation, (8) **sustainability** (including (9) resource efficiency, (10) protection of the environment, (11) social standards), (12) **digitalisation**, (13) miniaturisation, (14) new technologies, (15) internet of things, big data, (16) cooperation across industry sectors. Changing of

¹<https://www.zvei.org/presse-medien/publikationen/technologie-roadmap-next-generation/>

²<https://www.edacentrum.de/arbeitskreis-automotive>

legal boundary conditions will be considered. For the GENIAL! project, a team lead by Unity consulting³, researched key influencing factors, projected them into the future as scenarios, combined those scenarios to create new use cases and future scenarios and back projected them again to make graspable a farsighted and more complete picture of developments.

5.2 Ontology GENIAL! Modular Structure

This section explains the GENIAL! modular ontology suite in detail. It represents the status at the beginning of the project with its basic components that will eventually grow in sophistication. It was chosen to create a modular structure as the knowledge base of the project is expected to grow to considerable amounts. Some modules depend on others as well as some modules are relevant for some parts and use cases and not others. It may only be intended to share some modules with specific cooperating applications in order to provide input for the AgilA tool. The current basic components of the suite are described in the following and an overview is given in figure 5.2.

The overall ontology is structured in three main hierarchies. As top-level ontology BFO is used. On the second level is the GENIAL! Basic Ontology (GBO) with its fundamental classes, object properties and data properties. On the third level we have the two packages consisting of the context and roadmap information on the one hand and the complete car model on the other hand. As indicated in yellow we currently reuse BFO, the ontology of units and measure (OM) 2.0, ontology for innovation, the European materials modeling ontology, DABGEO external factors ontology, DABGEO populated places ontology, DABGEO weather ontology and the time ontology on various levels. We are developing the effect chain ontology, hw-sw allocation ontology, hardware software knowledge base with constraints, mission profiles ontology and e-Fuse ontology. The AI model, information security ontology and other roadmap ontologies are representative of the work in progress. Some of which are described in more detail in the next sections. The roadmap knowledge are for the key indicators and knowledge outlined in 5.1.2. The ontology documentation can be found online at⁴ and the OWL file at⁵. Additionally an github repository can be found at⁶. Several TelCos confirmed the potential applicability from experts. Below are the namespaces of mentioned components of the suite.

¹<http://purl.obolibrary.org/obo/bfo/2.0/>

²<http://www.ontology-of-units-of-measure.org/resource/om-2/>

³<http://www.purl.org/dabgeo/common-domain/environmentalfactors>

⁴http://www.purl.org/dabgeo/domain-task/smart_grid_scenario/building_district_city/populatedplaces

³www.unityconsulting.com

⁴<http://w3id.org/gbo>

⁵<https://github.com/wawrzik/GENIALOntologies/blob/master/OntologyModuleSuite/GENIAL!BasicOntology/GENIALOntBFO.owl>

⁶<https://github.com/wawrzik/GENIALOntologies/tree/master/OntologyModuleSuite>

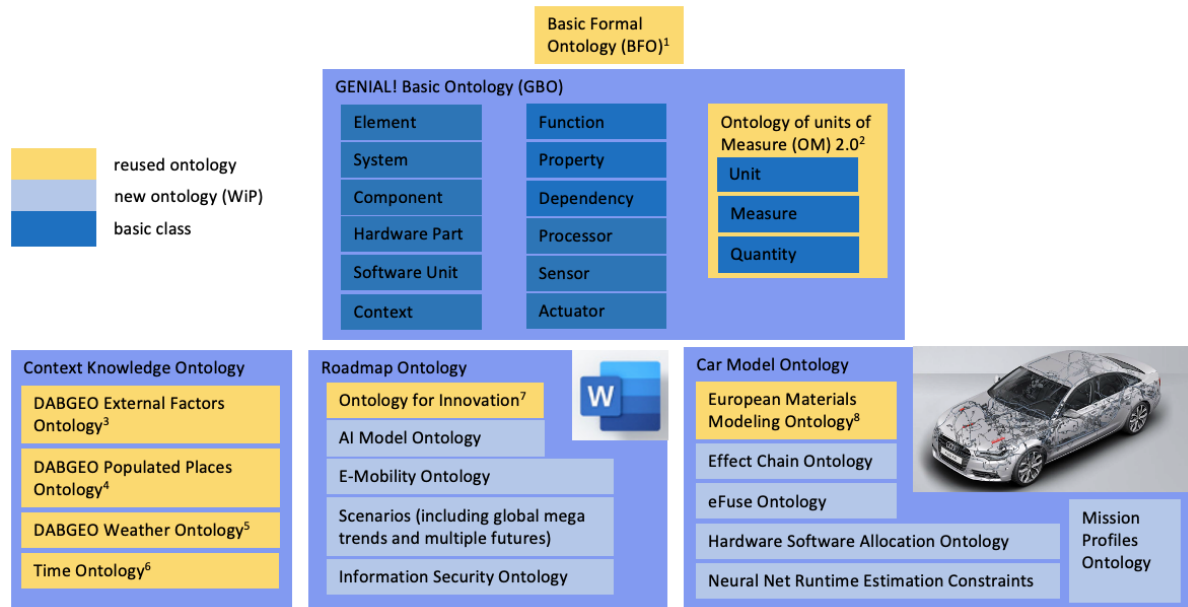
⁵<http://www.purl.org/dabgeo/common-domain/weather>⁶<https://www.w3.org/2006/time>⁷<http://purl.org/innovation/ns#>⁸<http://emmc.info/emmo>

Figure 5.2: GENIAL! Modular Ontology Suite

5.2.1 Basic Formal Ontology (BFO)

Basic Formal Ontology [ASS15] has been introduced and sourced in the biomedical domain, originating one of the most successful ontologies, namely the gene ontology. The gene ontology was so successful because it was able to describe genes consistently across wide domains of living organisms. It enabled interoperability between areas of interest developed independently. This success is now being transferred to the systems engineering and industrial domain by the IOF foundry [Kul+18]. Other benefits include:

- Common upper level for all needed classes, even those that will be added in the future
- The basic structure already had classes like *bfo:quality*, *bfo:function*, *bfo:object* and *bfo:role* that are fundamental to GBO and other classes
- Philosophical distinctions that help with dynamically changing, additional information

5.2.2 GENIAL! Basic Ontology (GBO)

The GENIAL! Basic Ontology (GBO) is designed to structure car components in a `has_part` hierarchy, attribute functions (e.g. the function to brake which can be executed partially by the motor or brakes), and properties to each component. Furthermore, properties can be modeled with dependencies. Dependencies are boolean or arithmetic expressions, which will in the end span the constraint net. Those dependencies are parsed and used to build a dependency tree in java as well as additionally in OWL. Basic concepts of the OM 2.0 ontology enhance the properties with their respective units. The context class is an additional class which is the superclass for context and roadmap information of the car model. This adapts the conceptualization of [CFM19] with BFO, as context is not seen as part of BFO, but a separate branch. In respect to BFO, Aristotelian definitions were used. Table 5.2 shows the basic vocabulary and its definitions in detail.

Compliance with ISO26262

"The ISO 26262 series of standards is the adaptation of IEC 61508 series of standards to address the sector specific needs of electrical and/or electronic (E/E) systems within road vehicles. This adaptation applies to all activities during the safety lifecycle of safety-related systems comprised of electrical, electronic and software components."⁷ This purpose and scope makes ISO26262 a good candidate to serve as a reference, definition and formalization for basic concepts of our ontology. We thus integrated the concept of component, hardware part and software unit into our ontology with its related concepts and aligned them with BFO. Figure 5.3 shows the formalization of component. To complete this definition we added 'element', 'non-system level element', 'system level element' and 'system'.

The definition of component according to ISO26262 is: **non-system level element** (3.41) that is logically or technically separable and is comprised of more than one **hardware part** (3.71) or one or more **software units** (3.159).

⁷<https://www.iso.org/obp/ui/#iso:std:iso:26262:-2:ed-2:v1:en>

GBO Classes	Definition
element	Continuant that is (according to ISO 26262) a system (3.163), components (3.21) (hardware or software), hardware parts (3.71), or software units (3.159)
system	Engineered object and system level element that is a (according to ISO 26262) set of components (3.21) or subsystems that relates at least a sensor, a controller and an actuator with one another
component	(According to ISO26262) (According to ISO26262) non-system level element (3.41) that is logically or technically separable and is comprised of more than one hardware part (3.71) or one or more software units (3.159). A component is a part of a system (3.163).
hardware part	a piece of hardware that is (according to ISO 26262) a portion of a hardware component (3.21) at the first level of hierarchical decomposition
software unit	a piece of software that is (according to ISO26262) an atomic level software component (3.157) of the software architecture (3.1) that can be subjected to stand-alone testing (3.169)
property	A quality or characteristic of a hardware part, software unit or function. A property has a boolean, integer, or real variable through the measure class.
function	A bfo:function that a component, hardware part or software unit implements
dependency	A gbo:property that is a boolean or arithmetic function / equation on properties. Used to be calculated with GENIAL constraint propagation. As string in LaTeX syntax
context	The circumstances that form the setting for an event, statement or idea and in terms of which it can be fully understood, e.g. Market, legal boundary conditions
unit	A bfo:quality that is any standard used for comparison in measurements
measure	A bfo:quality that are amounts of quantities
quantity	A gbo:property that is a representation of a quantifiable (standardised) aspect (such as length, mass, and time) of a phenomenon (e.g., a star, a molecule, or a food product). Quantities are classified according to similarity in their (implicit) metrological aspect, e.g. the length of my table and the length of my chair are both classified as length.

Table 5.2: Basic Vocabulary - Part 1

GBO Object Properties	Definition
has_part, part_of	Models hierarchical de-composition/integration, transitive
has_part_directly	Models hierarchical de-composition/integration of direct parts, intransitive
implements, is_implemented_by	Links elements to their functions and vice versa
executes, is_executed_by	Links processing units to the software on which they run and vice versa
depends_on	Links properties with dependencies
has_property, property_of	Links elements to their properties
has_value	Links properties or quantities to their measure (concrete value)
has_unit	Links measure with their unit
GBO Data Properties	
hasNumericalValue	Concrete value or digit of a measure

Table 5.3: Basic Vocabulary - Part 2



Description: component
<div>Equivalent To </div> <div>  non-system_level_element and (comprised_of some 'software unit' and comprised_of min 1 'software unit' or comprised_of some 'hardware part' and comprised_of min 2 'hardware part') </div>

Figure 5.3: Formalization of Component Definition

Aligning GBO with BFO

ISO 26262 could be made compatible with BFO by defining 'element' as an 'continuant', 'software unit' as 'generically dependent continuant' and 'hardware part' and 'system' as *subclass of* 'engineered object' which is a *subclass of* 'object' by using multiple inheritance. It is thus defined in a way that supports the automatic classification of the elements of our design into components if boundary conditions are met. To complete the definition of 'component', 'comprised_of' was defined as *equivalent to* 'has_part_directly' and 'non-system level element' *disjoint to* 'system level element'.

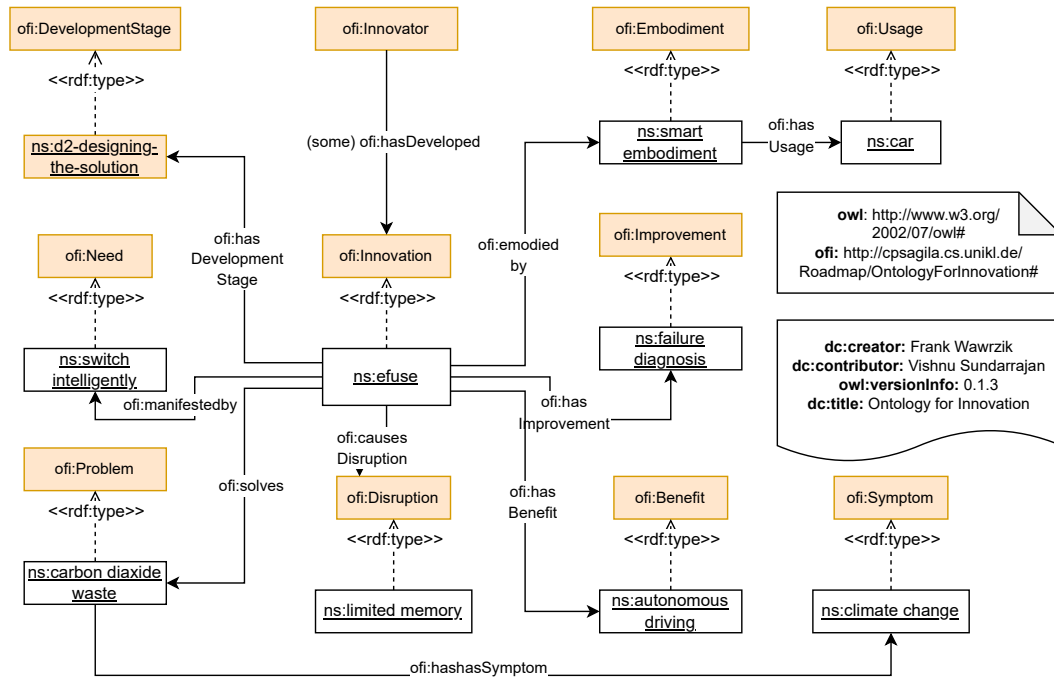


Figure 5.5: Instantiation of Ontology for Innovation for EFuse

fect chains.

As described earlier the effect chains are usually more informal, formalizing the knowledge of this stage is arduous work as the model is rather complex. Some trade-offs will have to be achieved or annotations may be used, because not always a formalization may be needed. Figure 5.6 shows the effect chain ontology with the central term 'interconnection'. Further some instance illustrate the example and instances of the effect chain are made disjoint with 'owl:AllDifferent' axiom. It is not part of the basic vocabulary but an enhanced vocabulary. Various effect kinds with its type can be modeled and related to properties and parts. Defining ports was omitted as in effect chains there are no real signals and some declarations might be considered as input/output or not.

5.2.3 EFuse Context Model with Solution Spaces

EFuse introduction

An electronic semiconductor fuse or eFuse is a fuse used in automobiles that additionally to a security function has the function of a relais. It switches more intelligently, yet yields other trade-offs. Various trade offs have to be considered when using and designing an efuse, especially assembly space, power dissipation and currents (among other constraints). Additionally to the constraints, context defines other solutions spaces and

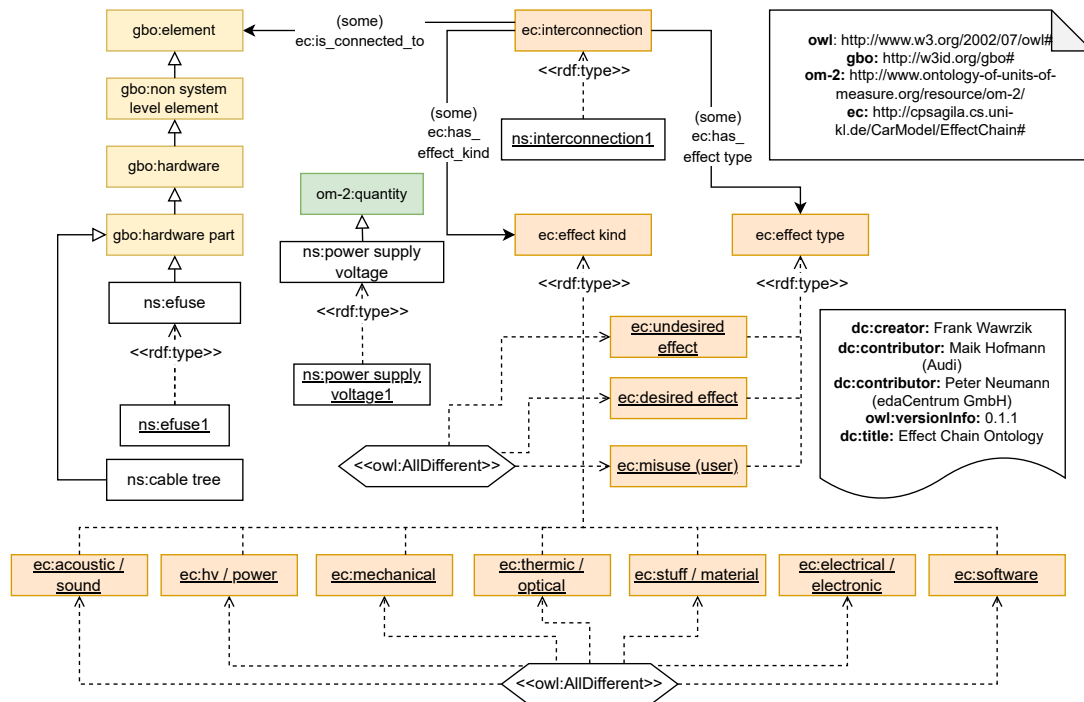


Figure 5.6: Effect Chain Ontology in GENIAL

affects constraints.

EFuse Context Knowledge

Relevant contexts (which are partly mixed with constraints) among others are:

- If a function is realized in hardware (and which hardware) or software
- Which materials are used, and what are their context trade-offs (e.g. drivers)
- Dimensioning of inrush currents
- Some transistors can measure current and thus diagnose
- There are many fuses in the car. Groupings into current amounts yields benefits
- Always-on/always-off technologies and secure vs. initial state (depends on application scenario)
- Which Tier 1 comes to which solutions

Solution Spaces

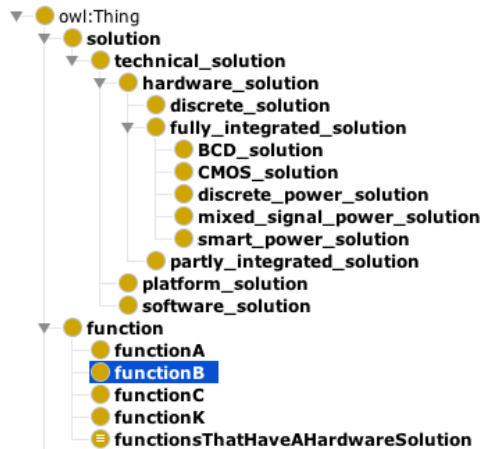


Figure 5.7: Class Hierarchy of the Solution Space of Hardware and Software Solutions

When going from an initial idea to an actual top-level requirement, decomposed requirements have to be applied to solution spaces to explore alternatives and their impacts. In this case each function/task has a solution space which get successively refined:

- Class hierarchy spans solution tree (Figure 5.7 shows an excerpt of the tree)
- Restrictions restrict solution tree for every function (Figure 5.8 shows refined space for functionB)
- Reasoner checks for consistency if restrictions are adhered to
- Keyword 'only' restricts solution spaces, keyword 'some' selects solution spaces

There seem to be several advantages to occupying the ontology with the solution space rather than having (just) plain java trees:

- Every solution space or part of a solution space exists independently
- Semantics are preserved which means reusability
- Additional rules can be defined over every solution space (inferred by defined classes)
- Reasoning
- Every solution space can inherit additional things/properties
- Other parts of the ontology can be combined with the solution spaces (indeed they are solution spaces as well)

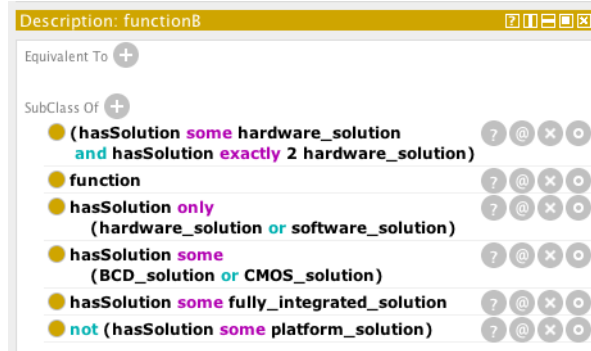


Figure 5.8: Restrictions on Solution Space of FunctionB

Excerpt from eFuse Model

Exchanging requirements along the value chain is a dynamic process, not only requiring ontologies and system models, but also a front end to make inputs and analysis. Figure 5.9 shows most information of a collection of eFuse related classes, restrictions and instances that were created while analyzing this process. It shows an incorporation of some parts and properties within GBO as shown in the top middle and right.

It also shows the instances containing also units, properties and measures as seen on the left bottom. The top left displays one possible solution for a smart power switch, here anonymously called solution4. Here we see the 'has_property' and 'implements' relation as restrictions on the class sol4_smart_power_switch. Notable is the cardinality restriction 'implements exactly 0 temperature gradient sensing', which negates that such a function exists. Also notable is the use of 'hasValue'-restrictions, which serve as a filler. They add relationships of specific instances to all instances of the class. This solution is how it is displayed in the frontend for users to edit.

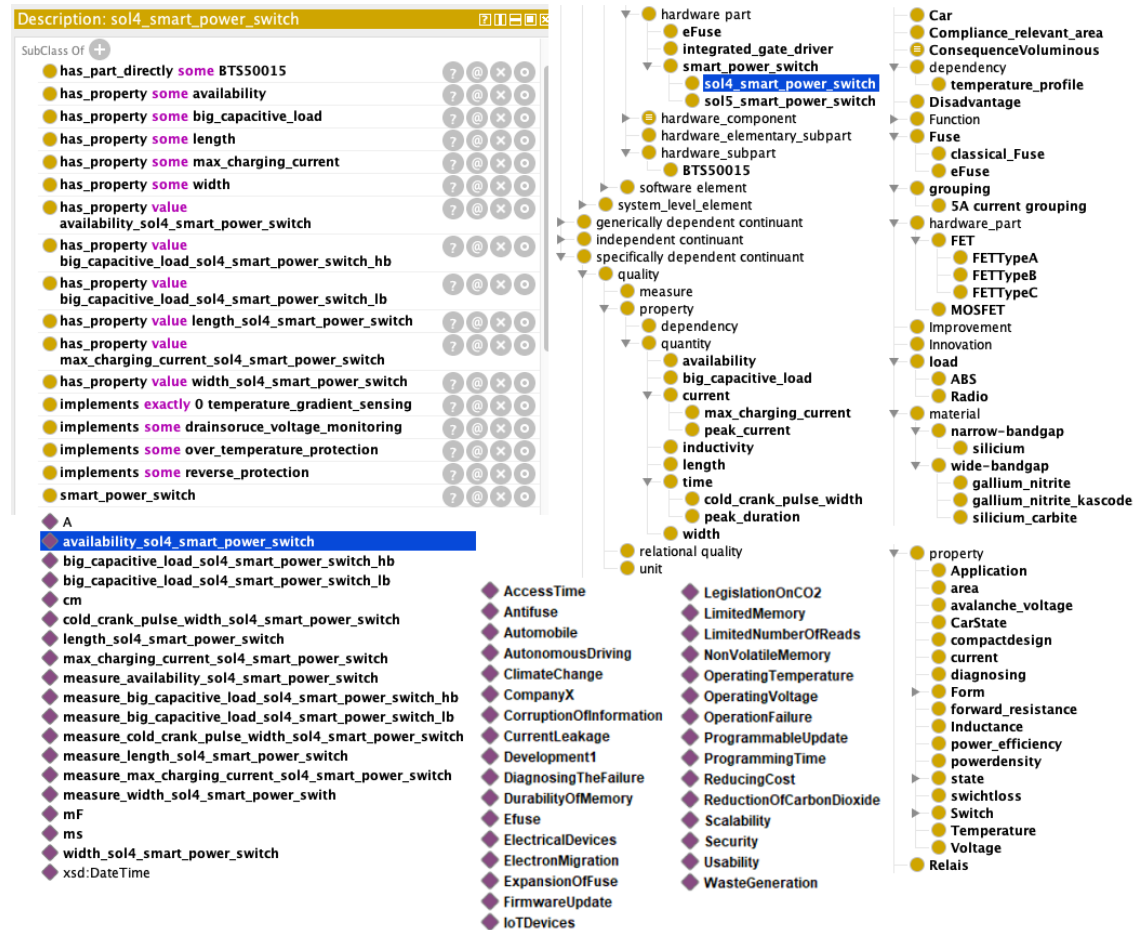


Figure 5.9: EFuse Prototype Showing Classes, Instances and Restrictions

SWRL Deductions

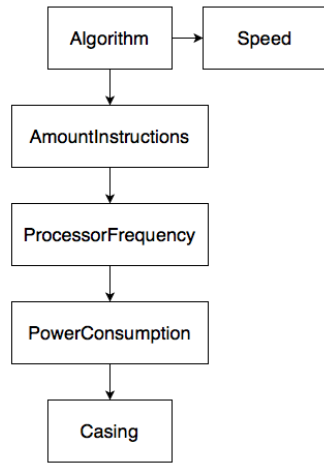


Figure 5.10: SWRL Rule Chain to induce new Casings

Next to properties of single components, the ascertainment of relationships is of importance. On the basis of these relations it will later be possible to determine the influences of local changes. As an example of refining a functionality (new algorithm), reduced processor instructions could follow. Which in turn could be used to lower the processor frequency, which reduces the power supply. This in turn enables a passive processor cooling, which influences the casing. Solving these deductions requires a combination of OWL and e.g. SWRL. Only through these formalisations of product and system properties a tool supported validation is possible. This allows to determine if a description is consistent or if a redesigned description actually is a refinement of a requirement. Figure 5.10 shows this rule chain and table 5.4 shows corresponding exemplary SWRL rules.

Rule Description	SWRL rule
An algorithm with speed 'average' has an amount of 150 instructions	Algorithm(?a) ^ hasSpeed(?a, average) → hasAmountInstructions (AmountInstructionsOldAlgorithm, 150)
An algorithm with the speed 'fast' has 30 instructions less	Algorithm(?a) ^ hasSpeed(?a, fast) ^ hasAmountInstructions (AmountInstructionsOldAlgorithm, ?y) ^ swrlb:subtract (?z, 150, 30) → hasAmountInstructions (AmountInstructionsNewAlgorithm, ?z)
If the amount of instructions is less than 125 the processor frequency can be reduced to 1.25 GHz	AmountInstructions(?x) ^ hasAmountInstructions(?x, ?y) ^ swrlb:lessThan (?y, 125) ^ ProcessorFrequency(?k) ^ swrlb:multiply (?z, ?y, 10) → ProcessorFrequency(processorX) ^ hasProcessorFrequency (processorXNew, ?z)
A processor frequency of less than 1.3 GHz leads to less power consumption	ProcessorFrequency(?x) ^ hasProcessorFrequency (?x, y?) ^ swrlb:lessThan (?y, 1300) ^ PowerConsumption(?k) → Processor (processorX) ^ hasPowerConsumption (processorX, low)
If a processor has less power consumption a new casing is the recommended action	processor (?x) ^ hasPowerConsumption(?x, low) ^ NewCasing(?y) → RecommendedAction(?y)

Table 5.4: SWRL Rules to Deduce New Casing from Faster Algorithm

Though the example is only illustrative, it shows untapped potential and new uses of ontologies and SWRL and combines qualitative and quantitative rule chains.

5.2.4 Modeling Hardware / Software Allocation for Processors (Prototype)

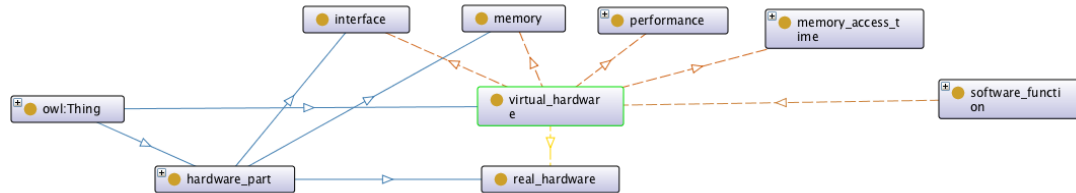


Figure 5.11: HW / SW Allocation Ontology Prototype

In GENIAL! we are considering possible not yet implemented parts and their solution spaces. We for example do not know if something is implemented in hardware or in software. Hardware-software allocation determines performance, memory size, memory access times and interfaces. It is thus part of the considerations of planning automotive components and part of the constraint net. We think a software unit or software function cannot exist without an underlying virtual hardware (processor, memory, etc.). And thus usually the mapping is not 1 to 1, but many software functions may be scheduled on one or more processors. The virtual hardware is then realized in concrete physical hardware parts. This conceptualization seems to address the hw-sw allocation problem and seems to fit with the idea of existential restrictions. With existential restrictions, we demand that there must be such a relationship between software and virtual hardware in order for a thing to be described completely and that it is conceptualized incorrectly if no such relationship exists. This prototype will be refined further to meet the use case in future works in cooperation with domain experts. The preliminary ontology can be found in figure 5.11.

The virtual hardware is in the center and is distributed to min 1 real hardware (*gbo:hardware part*). Further it has the requirements interface (*gbo:hardware part*), memory access time (*gbo:property*), memory (*gbo:hardware part*) and performance (*gbo:property*). The software function is a function (*gbo:function*) which runs on exactly 1 virtual hardware. In a next complex step the virtual hardware gets scheduled on a real hardware, which requires a selection of variants.

5.3 Formalizing Roadmap Knowledge

5.3.1 Starting with some Initial Models

This subsection refers to work done in collaboration with a master thesis entitled 'Towards a true microelectronic and automotive digital roadmap' [Sun20]. Aim of the thesis

was to explore how roadmap knowledge as described above can be modeled and create first models of such domains.

eFuse Roadmap, AI and InfoSec Ontology

The properties and structure of an efuse are highly influenced by roadmap decisions. The student traversed basically an UML model into OWL with some changes. This model contains influence chains linking a concrete model and hardware parts to abstract influences like climate change. And through the formalization of this knowledge we can then estimate e.g. if a law changes (or we find out the cause of climate changes is initiated by solar magnetic surface cycles of the sun rather than CO2 emissions) that there is an actual influence of properties directly in the model (or at least that future versions have to be modelled according to new roadmap information).

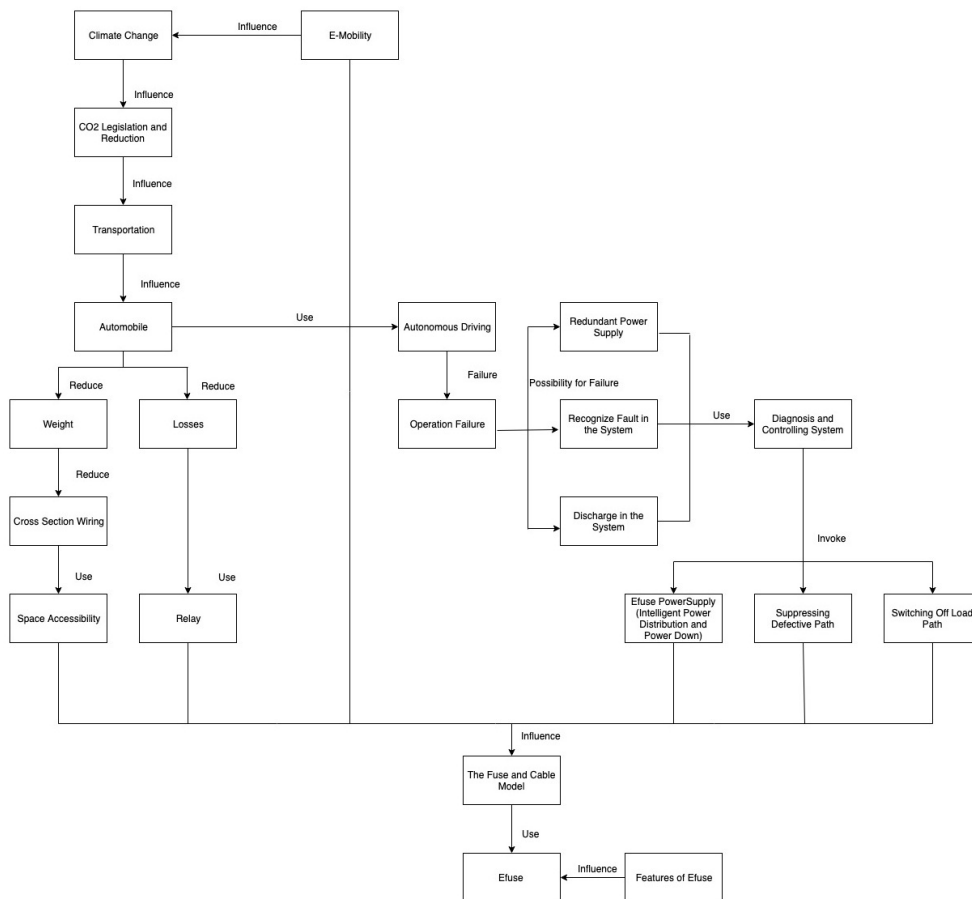


Figure 5.12: Electric Fuse Ontology Model [Sun20]

The student describes figure 5.12 as follows: [Sun20] 'In the cable model diagram,

the diagram mainly focuses on climate change, autonomous driving, and e-mobility. In climate change, the cable model deals with carbon dioxide reduction and legislation aspects. Using the cable model in the transportation domain, how can we impact the reduction of carbon dioxide. The cable model provides us with more information on how the reduction of carbon dioxide can be achieved. In the automobile, vehicles used by both public and private transportation can help us achieve the mitigation of carbon dioxide. Also, reducing weight leads to the reduction of carbon dioxide. The weight reduction is supported by reducing cross-section wiring. The reduction in cross-section wiring directs to the space accessibility. The solution for all the problems from weight reduction, reduction of cross-section wiring, and ultimately reducing carbon dioxide in the automobile is using efuse. An electric fuse can recognize the fault and provides a quicker response to the fault and recover from it. Also, the electric fuse has abilities such as protecting the system from surge and have maximum fault current. The location of the electric fuse does not have any restrictions like the thermal fuse, whereas thermal fuse is located in a fuse box. This is one of the reasons that help to reduce the weight in the automobile. The reduction weight indirectly reduces the cost implied as well. Let us see further how to reduce losses in the automobile. Power distribution is challenging in the automobile, using relays and fuse is one of the solutions, which can reduce the losses in the automobile. Going from mechanical to semiconductor relays reduces weight and cost. [Dup17; JB17]

The autonomous driving is another focus area of the cable diagram. In autonomous driving, the operation in case of failure is considered and how the operation failure issues is addressed with the help of the cable diagram. The solution to the problem in autonomous driving is the electric fuse and cable model. The shortlisted reasons for the operation in case of failure are redundant power supply, discharge onboard system, and recognizing the fault in the onboard system. These shortlisted reasons need a diagnosis and control for the onboard system and onboard system management. There are few actions required for diagnosing the fault in the onboard system, controlling the fault in the onboard system, managing the onboard system as well. The actions are suppressing the defective paths, intelligent power supply, power down, and switching off the path loads.’

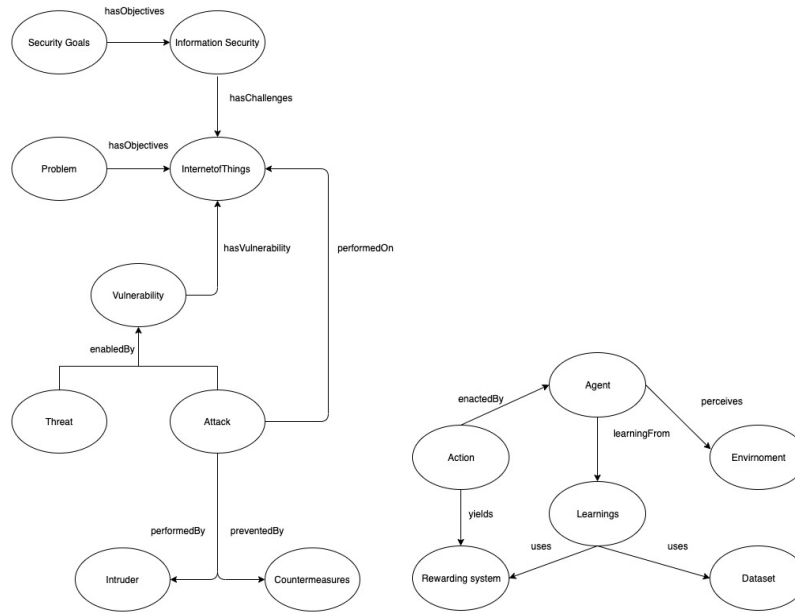


Figure 5.13: Summary of InfoSec (left) and Artificial Intelligence (right) Ontology [Sun20]

Figure 5.13 shows the result and an excerpt of both the information security and artificial intelligence ontologies. For more information see [Sun20].

5.3.2 Further Models

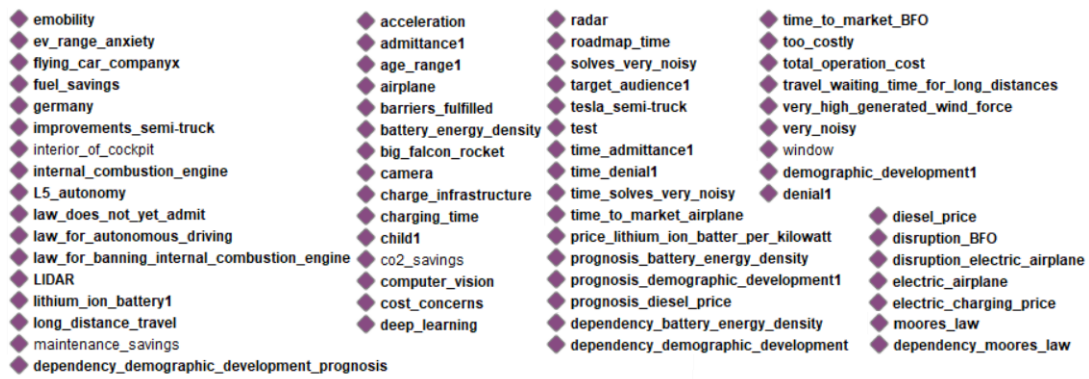


Figure 5.14: Individuals of the Roadmap Ontology

This section builds on the previous one in that it further describes such knowledge in areas such as legislation/laws, demographic development, innovations, e-mobility, autonomous driving, but also gbo:hardware parts and gbo:properties and some rules. Figure 5.14 shows the individuals in the complete roadmap ontology.

Demographic Development

The question may arise if an ontology is the 'right place' for information like demographics. However as can be seen in the following, it is often the case that some information needs semantics whereas other information might rely on mathematics, which can then be both combined. Figure 5.15 shows existential restrictions on the class demographic development and the triples of a specific demographic_development1 instance. Some classes define context like for example country. Each demographic allows then to be loaded contextually from where you log in. The dependency class links the demographic to its mathematical curve or table. Furthermore each demographic is linked to its prognosis which contains a future estimated model of the demographic development. Each demographic is also linked to a target audience (which is a semantified specific age range) and an age range, which allows to e.g. select relevant parts of the population.

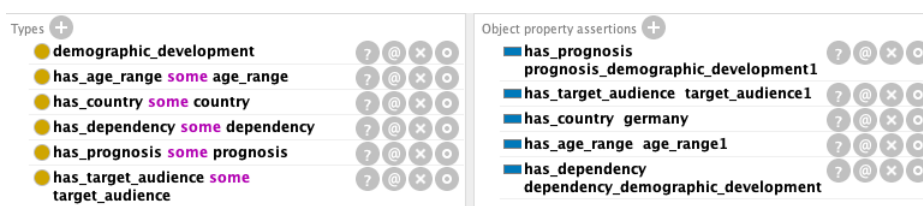


Figure 5.15: Demographic Development Ontology Model

Legislation and Laws

Exemplified are two laws:

1. law for allowing autonomous driving
2. law for banning internal combustion engines

Though a law for allowing autonomous driving seemed to be a complicated issue, soon solutions appeared (e.g. driver has to touch the wheel at certain times) in the U.S. in the advent of Tesla. Yet a law can be seen as something disruptive which can easily and quickly prevent a technology. It is thus part of a roadmap and complicated intersections. The law for banning internal combustion engines can be viewed critical in that it intervenes in the freedom of the people and superimposes certain values. It is assumed now that the electrical car is superior with at the same time competitive price and may naturally replace it.

A legislation/law is conceptualised as something which has a 'consequence', which can either be an 'admittance' or a 'denial' of something, e.g. an innovation. A 'consequence' in turn has a point in time in which it occurs.

Innovation, Improvement, Disruption and Barriers

The Ontology of Innovation as introduced earlier contains the classes 'innovation', 'disruption' and 'improvement'. Three classes which have been reused here. As Tesla is currently leading the market and is highly innovating in several areas in a disruptive way, some of their products have been modeled here. The disruptive character of those innovations make other innovations obsolete or improves them in specific ways. One example is the 'Tesla semi truck' which has the improvements of 'co2 savings', 'interior of cockpit', 'maintenance savings', 'fuel savings', 'acceleration', 'total operation cost' and 'window'. Other examples might be a flying car company that now solved the problem addressed by Elon Musk about flying cars. Namely, that the issue of 'very noisy' has been solved to the degree of motorcycle level noise. Or the big falcon rocket, which 'deprecates' the conventional airplane and 'liberates' 'travel time for long distances'. The words in apostrophe (') are elements of the ontology (classes, instances, object properties). Furthermore there are 'barrier's like 'e-mobility barriers' or 'flying car barriers' or 'tunnel barriers' (Elon Musk builds tunnels for autonomous vehicles to avoid traffic and go with faster speed). E-mobility barriers are currently 'cost concerns' as the EV's are still very expensive (this is supposed to be solved with scaling production and improved technology in a few years). Others are 'charging time', 'ev range anxiety' and 'charge infrastructure'.

Properties, Explanation, Recommended Action and Rules

Example of properties in the roadmap are 'times' like 'time to market' or 'roadmap time'. 'carbon footprint', 'battery energy density', 'filling station density' (rather than e.g. gas station, because it could be anything like electricity or hydrogen). 'autonomy', 'affordability' or 'age range'.

It might be useful that the ontology yields 'explanations' for what is going on. E.g. as part of shifting the roadmap time, barriers might be fulfilled ('barriers fulfilled'). But a law might not yet admit it ('law does not yet admit').

Furthermore we have the 'recommended action' e.g. 'don't do this / not recommended' or 'execute this action'. Figure 5.5 shows a few first rules.

Rule Description	SWRL rule
1.If the battery energy density gets over 400, then the electric airplane becomes feasible	uo:battery_energy_density(?bed) ^ uo:has_numerical_value(?bed, ?bednm) ^ swrlb:greaterThan(?bednm, 400) → uo:feasible_innovation(uo:electric_airplane)
2.If roadmap time greater ttm for e-mobility barriers, then emobility becomes a feasible innovation	uo:roadmap_time(?rt) ^ uo:has_numerical_value(?rt, ?dprt) ^ uo:e-mobility_barriers(?barr) ^ uo:has_ttm(?barr, ?barrttm) ^ swrlb:greaterThanOrEqual(?dprt, ?barrttm) → uo:feasible_innovation(uo:emobility)
3.If airplane is a design and has ttm greater than ttm BFR (Big Falcon Rocket) AND has purpose long time travel, then dont do it (the design)	uo:design(uo:airplane) ^ uo:Disruption(?disr) ^ uo:deprecates(?disr, uo:airplane) ^ uo:time_to_market(?ttm) ^ uo:has_numerical_value(?ttm, 2028) ^ uo:has_property(?inno, ?ttm) ^ uo:Innovation(?inno) ^ has_purpose(?inno, long_time_travel) → uo:dont_do_it(uo:airplane)
4.If a barrier is solved by an innovation at roadmap time it becomes a solved barrier	uo:Innovation(?inno) ^ uo:solving(?solv) ^ uo:solves(?inno, ?solv) ^ uo:has_time(?solv, ?solvtime) ^ uo:has_numerical_value(?solvtime, ?slvtime) ^ uo:barriers(?barr) ^ uo:solves(?inno, ?barr) ^ uo:has_numerical_value(uo:roadmap_time, ?rmpt) ^ swrlb:greaterThan(?rmpt, ?slvtime) → uo:solved_barriers(?barr)

Table 5.5: SWRL Rules for Roadmapping

As can be seen, rule number 2 could be improved. I.e. it is not that as soon as one of the e-mobility barriers is fulfilled, that e-mobility will be a feasible innovation. But rather that if ALL the barriers are fulfilled, it is feasible. A feature which is lacking in SWRL, but can be integrated.

Rule number 3 describes a direct use case when writing a roadmap or instantiating a design (which might acquire the information from context). The roadmap is integrated in the tool and is intertwined. A notification can pop up, indicating that the very innovation may be disrupted and maybe yielding what has to be improved to be competitive. Note that this is just an example as well, exploring in what a digital roadmap yields advantages over formal ones.

In rule 4 a method is applied that is known from upper ontologies. 'solving' is a class that would usually have been an object property. This way it becomes possible to add a time of happening to a verb. Which is needed in this case.

5.4 Hardware Domain

As a proof-of-concept in the following a hardware domain is modelled. It shows a utilization of the GENIAL! Basic Ontology and an application of the concept. First, we fill the knowledge base with some basic hardware concepts and instances that are part of the simulation models of University of Tübingen. They build hardware and software models of controller architectures with a hardware/software mapping in order to make performance estimates and reduce energy. These are also crucial in the deployment of automotive hardware and software.

In the following example a "wake up" function is deployed in a car that uses a neural net for speech recognition. The architecture for computation consists of a general purpose processor and dedicated hardware. The following two pages give an overview of the hardware architecture.

5.4.1 Evaluation

Figure 5.17 and figure 5.16 show an excerpt of the ontology and knowledge base. In general most of the modelling was straightforward and some modelling did take more consideration. The ISO26262 was sufficient to express the model. The result was a five level hierarchical decomposition ranging from the system to the components (hardware or software) to the hardware parts/software units and two levels of the hardware sub-parts. The hardware elementary subparts were too detailed for the simulation level. The definitions held in the ontology assured correctness of the hierarchy and its levels of both the ABox and TBox. The has parts hierarchy was more strictly modelled with the universal restrictions in order to ensure the hierarchies correctness. The result also showed what might be a part of the ontology and the knowledge base and what might better be kept in UML or both. The classification of processors e.g. as 'general purpose unit' or 'custom purpose unit' and memory types like 'internal memory', 'volatile memory', 'RAM', 'SRAM', etc. are good candidates for the ontology. As well as the classification in the basic ontology from ISO like hardware part.

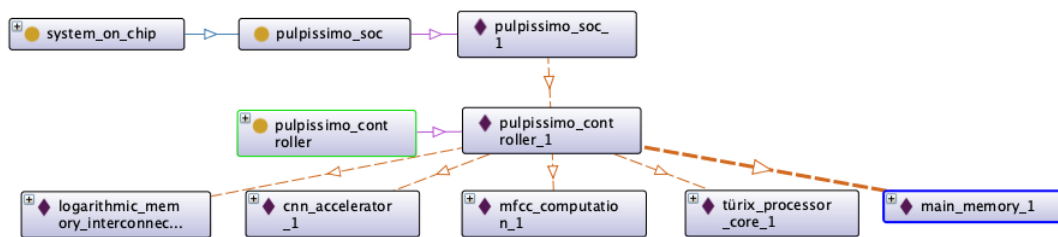


Figure 5.16: Instance Hierarchy of Hardware and Software Elements

The actual model can be part of both languages, i.e. performance relevant properties in the ontology and the UML model. The parts can be part of both SysML/UML and

OWL as well in case they have proper semantics. Figure 5.16 gives a short overview of the first three hierarchies of the instance model. The reasoner computes transitivity in the has-parts hierarchy and supports navigating through the hierarchy. Creating logical definitions can support consistency but also hinder possibilities. E.g. a system bus which is a bus cannot consist of other buses like a address bus, a data bus or a address bus. This would be in conflict with the ISO definition that the next hierarchical level of a bus (because it is a hardware part) must be an hardware subpart (and not a bus again in itself).

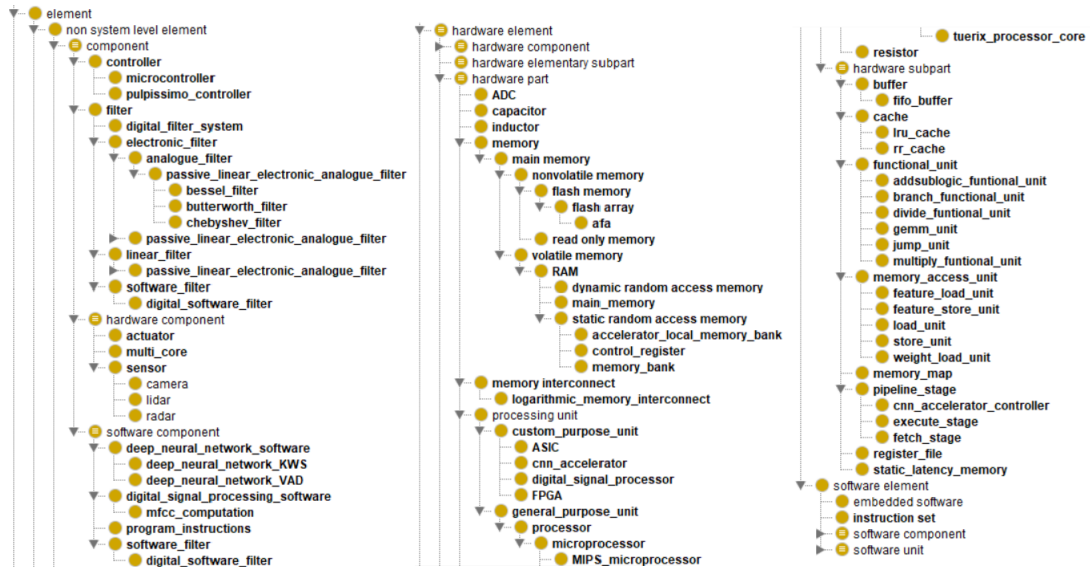
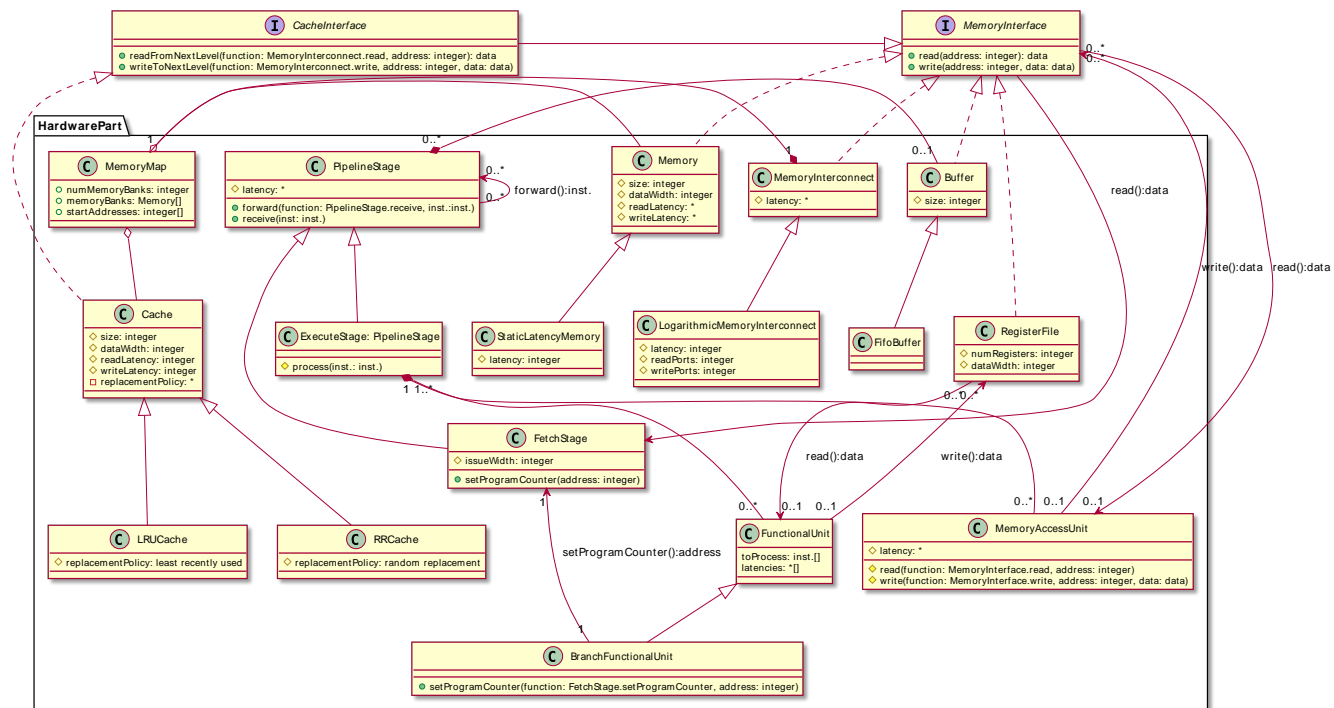


Figure 5.17: Taxonomy of Hardware Elements



Constraints The following shows a summary of the taken constraints and formulas that are used to predict a cycle accurate runtime of a neural net. The constraints are taken and explained in [Ber+20]. Explanations are omitted for compactness, but basically parameters like the input channel width, input channel number, output channel number, filter width and stride determine performance.

$$\hat{C}_w = C_w + p \cdot 2 \cdot \left\lfloor \frac{F}{2} \right\rfloor \quad (5.1)$$

$$a_w = \left\lfloor \frac{\hat{C}_w - F}{s} + 1 \right\rfloor \quad (5.2)$$

$$C_{w,b} = \left\lfloor \frac{F}{2} \right\rfloor \quad (5.3)$$

$$a_{p,b} = p \cdot \left\lfloor \frac{C_{w,b} - 1}{s} + 1 \right\rfloor \quad (5.4)$$

$$\#MAC_{not,b} = \sum_{i=0}^{a_{p,b}-1} \left\lfloor \frac{F}{2} \right\rfloor - s \cdot i \quad (5.5)$$

$$F_w = a_w \cdot s + F - s \quad (5.6)$$

$$C_{w,e} = F_w - C_w - C_{w,b} \quad (5.7)$$

$$a_{p,e} = p \cdot \left\lfloor \frac{C_{w,e} - 1}{s} + 1 \right\rfloor \quad (5.8)$$

$$\#MAC_{not,e} = \sum_{i=0}^{a_{p,e}-1} \left\lfloor \frac{F}{2} \right\rfloor - s \cdot i - (C_{w,b} - C_{w,e}) \quad (5.9)$$

$$t(l) = 1 + \left\lceil \frac{C}{8} \right\rceil \cdot \left\lceil \frac{K}{8} \right\rceil \cdot (a_w \cdot F - \#MAC_{not,b} - \#MAC_{not,e}) \quad (5.10)$$

$$T(\mathbb{L}) = \sum_{l \in \mathbb{L}} t(l) \quad (5.11)$$

5.4.2 Application

We started building our hardware / software library with an ontology of a digital twin of a water level monitor [SGW21], added simulation models of cyber-physical systems [Waw+15] and the neural net accelerator of an UltraTrail processor. For the neural net use case, we used both, a self-developed language for properties and dependencies, called APPEL [Gri+21] and a knowledge base in OWL and ArangoDB. Whereas APPEL

targets more the acquisition of components, functions, properties and dependencies for non-domain experts, the knowledge base and ontology refine the acquired knowledge, which is then further classified and structured. Both are interoperable.

This use case exemplifies the building of a components library and its interaction with roadmap constraints. For that a neural net constraints model is coupled with a car model and its roadmap constraints. For the car model we have a simple model that contains electric cars and combustion engine cars with its mass structure and various other properties, I call "values", which can represent any kind of property or an immaterial, exemplary value. The model is structure-aware and propagates its constraints through its parts, here implemented as SUM function. Below is the APPEL model with parts and constraints of the electric car:

```
ElectricCar isA Car
  it hasA wheel : Int (4..4) Wheel
  it hasA body1 : Int (1..1) Body
  it hasA chassisp1 : Int (1..1) Chassis
  it hasA chassisp2 : Int (1..1) Chassis
  it hasA battery : Int (1..1) Battery
  it hasA evpowertrain1 : Int (1..1) EVPowertrain
  it hasProperty ElectricalCarMass : Real =
    SUM(mass)
```

In the following Moore's Law and others are formulated as constraints. A prediction of the evolving of battery prices after a McKinsey study and inflation are combined. This yields predicted prices at time1 in tinyyears. Note how dependent properties are imported with the depends_on relation.

```
MooreLaw isA Context
  it dependsOn Library.UltraTrail
  it dependsOn Library.time1
  it dependsOn Library.DeepNeuralNetworkLayer
  it hasProperty runtimein : Real = t_1
  it hasProperty runtimeout : Real =
    runtimein/power2(0.5*tinyyears)
```

```
Battery isA MechanicalObject
  it dependsOn Library.time1
  it dependsOn Library.Inflation
  it hasProperty startvalue : Real
    (14000.0..14000.0)
  it hasProperty price : Real =
    startprice*(powerb(0.94408, tinyyears))
```

These kind of formulations now enable us to explore relationships and correlations that are not obvious and yield new viewpoints and enable new technologies. For example we can see that in 5 years the battery values, like battery density or other "values" will drop in a way that BEV's will be more lucrative than ICE cars. We can then explore for example that the new runtime of our neural net on a similar processor can allow us to increase the filter width or number of input channels in a way that increases the

accuracy of our neural net or yields a new application. Furthermore all parameters are in ranges and its dependencies stored via AADD. This enables analyzing the amount of total uncertainties and its factors and where they contribute the most to the parameter range.

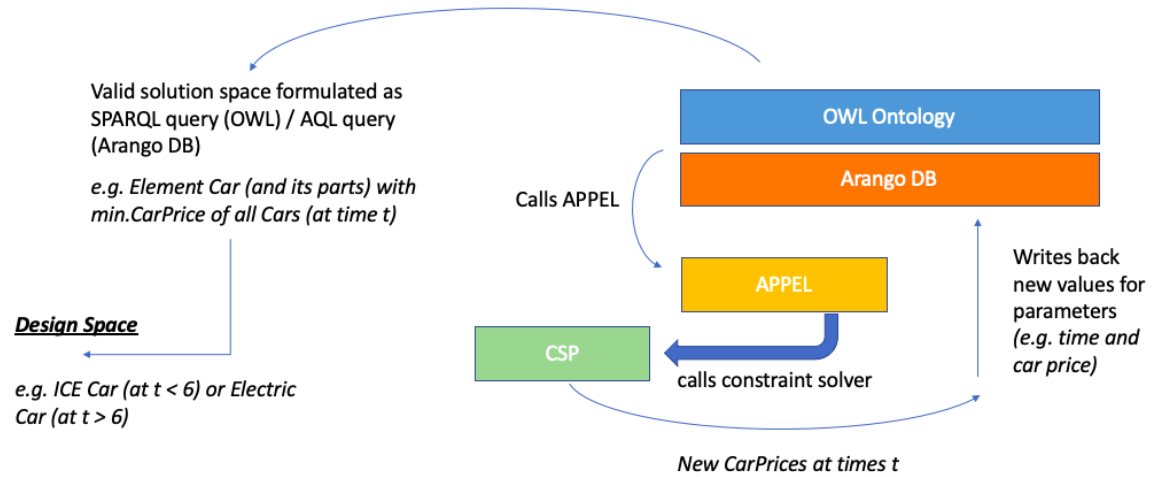


Figure 5.18: Interaction of Constraints and Knowledge Base

Figure 5.18 shows the overall implementation and interaction of constraints, ontology and database.

We distinguish

- *the ontology part*, which yields the reference model with its definitions. It also checks for consistency of the APPEL model via reasoning
- *the database part*, which holds the triple load, the relationships and the constraints
- *the APPEL model*, which holds the constraints in a user friendly way and the parts and elements needed for calculating the constraints and then calls the constraint solver to update the variables
- *a design*, which is a solution space over a knowledge base that fulfills the constraints of the design

5.4.3 Further Development of APPEL

In a second version, APPEL was advanced into the Markdown language and renamed as SysMD. This enables embedding of constraints and triple expressions, next to actual natural language expressions and using constraints, models and text together. This syntax slightly evolved and semantics, meta model and commands were combined. This is now a true 'living document' and executable with code. Figure 5.19 gives an impression

of some dependencies. There are packages, with a root and a global package. 'Defines' defines a class in a package and we introduced a more general 'hasA' object property. Further, we see 4 variables with ranges, that are calculated with a constraint in 'a' and display those with the 'Display' command. Units are adapted and internally calculated. Note the above text before the actual code or knowledge base, which is like a normal web page or Word document.

Simple dependencies

Dependencies between properties can be modeled by using boolean or arithmetic expressions.

```

1 Dependencies isA Package.
2
3 Dependencies defines component isA Component.
4
5 Dependencies::component hasA
6
7   d: Real(10.0 .. 20.0)[mm],
8
9   c: Real(2.0 .. 3.0)[m],
10
11  b: Real(1.0 .. 2.0)[km],
12
13  a: Real = b*c*d.
14
15 Display hasA 'component we have the following properties': Dependencies::component
16
17 In component we have the following properties:
18 d = 10..20 mm
19 c = 2..3 m
20 b = 1..2 km
21 a = 20..120 m^3

```

Figure 5.19: SysMD Dependencies Example in Tutorial

Chapter 6

Conclusion and Outlook

Developing ontologies is an intricate task and the future will bring much innovation to this field. This work has presented ontologies and showed some applications related to the engineering 4.0 domain.

It started by both utilizing SysML and OWL as prior work (as shown in the appendix). With SysML a design methodology was introduced that allowed to simulate models with power and uncertainty properties across different domains. A comparison of both languages gave context to the work that followed.

From that point it was decided to deepen/branch to ontological engineering rather than systems engineering as a matter of interest. Foundations have been researched and presented to get an accurate understanding of the technology. This entailed reasoning, language foundations, categories of ontologies, state-of-the-art ontologies and design principles and best practices for ontology design. Further advanced concepts were introduced like patterns and antipatterns, multi-level theory or foundational ontologies from other research.

A prototype tool with a graphical user interface was built that allowed for accessing the knowledge base, writing to it and calculating constraints with its observed limitations. A preliminary expert system was built and integrated with the ontology. This helped kickstarting the project and served as a starting point for further developments. For building ontology models the GENIAL! ontology modular suite was created. It contains a basic vocabulary that defines the meaning of the defined classes. For car models this entails classes like element, component, system, hardware part, software unit, dependency, property and others. The vocabulary is consistent with ISO26262 and BFO. A hardware software domain ontology and knowledge base was designed, as well as eFuse, roadmap, effect chain, context and innovation ontology prototypes and other models.

Further it was explored how workpackage 1 could be integrated with workpackage 3 by creating models of roadmap information domains. And finally the whole approach was demonstrated with an application framework based on Spring, REST, ArangoDB, OWL, OWL Reasoners and symbolic constraints and demonstrated with an example of roadmap and hardware / software constraints. GBO was well axiomatized to support active reasoning and a precise computerized definition.

The project is now in year two and requirements have finally been elicited. The designed ontology suite can now be applied and new modules can be created as well as the library can be built and filled with information. This entails the sensor domain in general, mission profiles, hardware processors, further roadmaps and much more. The ground work has been necessary to develop good models and know how and when to use them. The approach allows a coupling with existing IoT infrastructures.

6.1 Outlook

This section describes some follow-up research directions that seem rewarding.

Context and Ontology Assume that an A.I reads a text and concludes that Person X is a virtuous individual. And only virtuous individuals may have access to certain things. How are these conclusions drawn? How are they subject to change? Should this be done? What is the context for a certain action to take place? What is the context that legal systems or legal laws change or operate? This may include building stratified levels of context. Using various context classes and merging them together depending on context. And linking triples to context instances. It is evident that context will play a major role in the internet of things and knowledge-based system.

Security and Ontology In the area of ontology, security takes on again a different significance. It is one thing when your data is stolen, quite another when the meaning of your data is stolen with it.

Language Foundations There are a variety of different ontology languages (e.g. F-Logic, SROIQ, CL, OBO) and there is different expressiveness of OWL. Understanding how a language comes about and how it enables achieving its means is an essential part of further developing concepts that help in achieving a unified logical and contextual layer.

Machine Learning and Ontology (Semantic Machine Learning) Is an emerging field, which has become a hot topic. [LS19] discusses some drawbacks of dNN's (deep neural networks) for AI in general and how semantics can provide complementation. As machine learning has become part of many applications, semantics is needed to effectively share, represent and access the generated results.

Web of Things Stack to support the internet of things. Rewarding area for ontology builders and most thriving field in ontology development at this time.

Foundational Ontologies FO's have shown to be useful in many ways to approach thinking about ontological issues and support better solutions. Beginning with the theories of Aristotle they provide a sound basis for forming ontological models.

Web of Thought An example could be capturing intentions (e.g. through brain waves) and then actuating on them when evaluated as firm. E.g. the intention to go somewhere and then in addition actuating/opening the doors that are in ones way when those doors are in proximity. Very sensitive area that requires careful proceeding.

Higher Levels of the Semantic Web Stack How can we trust certain information and what is the proof for this trust? How can we build ethical A.I. systems? The author did find very little information on how to achieve a unifying logic. How can we prevent

trust being misused?

Unification (*Global vs specific knowledge space*) There are many heterogeneous models even with ontologies, which are actually supposed to solve the interoperability problem, rather than contribute to it. Unification is an open issue that needs to be addressed. Global information and local information play together, are shared on different levels, involve different levels of abstraction and need to be connected.

Reasoning and Ontology By doing research, it was peculiar that few reasoning was used in the applications (mostly used for interoperability). It may be rewarding to make use of it more and find its benefits and limitations.

New Applications and Ontology Heavy use in the internet of things, less in other areas. Finding new use cases and applying it in different ways offers much potential to be harnessed. As it is supposed to be the backbone of the internet, much awaits.

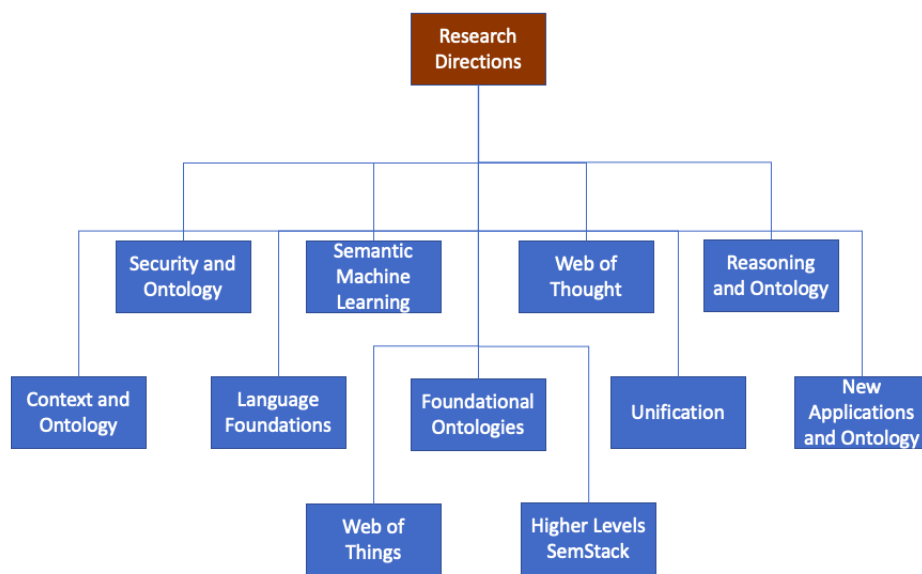


Figure 6.1: Research Directions Outlook

Chapter A

Appendix

A.1 More Complete Reference of Ontologies

This section gives a more detailed overview. Axioms between the classes are shown and a more unified view of how the different parts are connected. Due to its size, only the most important parts of the suite are outline: BFO, GBO and HW / SW Domain. Finally, the complete APPEL model of the application use case is attached and the content of the database, which includes the HW /SW knowledge base is visualized.

A.1.1 Abstract and Overview

Prefix	Namespace
bfo	http://purl.obolibrary.org/obo/
gbo	http://w3id.org/gbo#
sosa	http://www.w3.org/ns/sosa/
om-2	http://www.ontology-of-units-of-measure.org/resource/om-2/
hardware	http://cpsagila.cs.uni-kl.de/GENIALOnt/hardware/
hardwareKB	http://cpsagila.cs.uni-kl.de/GENIALOnt/hardwareKB/
dc	http://purl.org/dc/elements/1.1/
dcterms	http://purl.org/dc/terms/
owl	http://www.w3.org/2002/07/owl#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#

Table A.1: Prefixes and Namespaces used within my Neural Net Accelerator Knowledge Base

Ontologies and knowledge base are implemented using OWL-DL 2. According to 3.1.3 GBO can be classified as a middle-level core ontology.

To show the (potential) application of GBO, a reasoning use case for classification is shown. Further an instantiation example is given. GBO as well as the component of the module suite are supposed to be constantly evolving. Thus, after wide spread application in automotive, even substantial revisions of GBO are possible and likely.

The namespaces with its prefixes that are referred to in this appendix are seen in table A.1.

A.1.2 Basic Formal Ontology

BFO as top-level describes the overall structure on how to classify and partition the classes. It is a realist ontology and designed for practicality. Some of its parts were outlined in section 3. Figure A.1 shows all classes, also of the occurents.

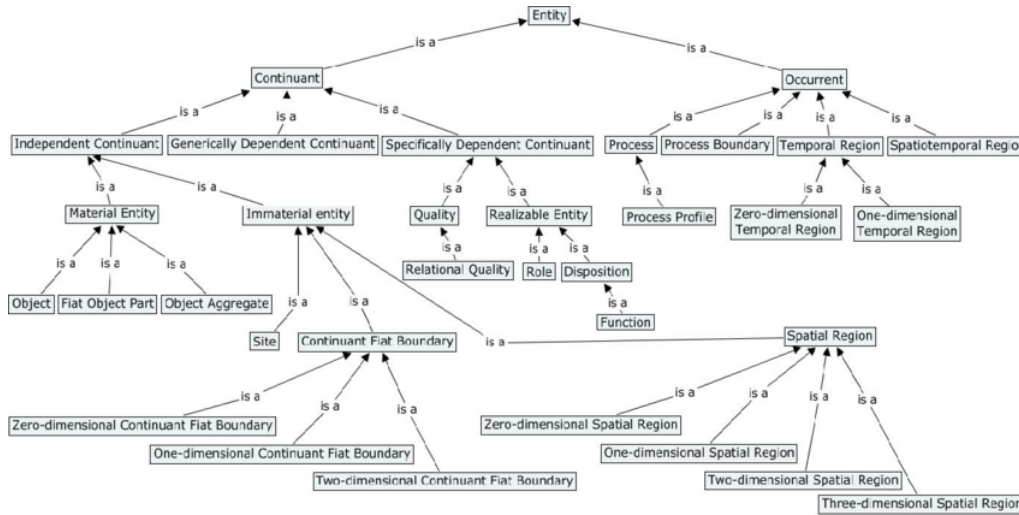


Figure A.1: BFO 2.0 All Classes [Abh16]

A.1.3 GENIAL! Basic Ontology

GBO is designed to facilitate the exchange of microelectronic components along the automotive value chain. It is intended to be imported by any OEM, Tier1, Tier2, semiconductor manufacturer or suppliers. And its purpose can also be beyond that and be used to describe microelectronic technology in general. In order to do that,

1. It is based on Basic Formal Ontology (<https://basic-formal-ontology.org/>)
2. It is based on ISO 26262 standard and definitions (<https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en>)
3. Integrated into AGILA / SysMD framework with client-server architecture

It can describe the context of a component and its influential factors as well as the hardware, software or other mechanical parts of the cars and its properties.

To model hardware and systems it contains classes like:

- hardware, element, system, item, hardware part, hardware subpart etc.

To describe software it contains classes like:

- software, software element, software component, software unit

To describe related properties or functions it contains classes like:

- function, property, dependency, quantity, unit, measure, etc.

For modelling relationships it currently contains 14 object properties:

- comprised_of, has_part_directly, has_part
- depends_on
- executes, is_executed_by
- has_unit
- has_value
- has_property, property_of
- implements, is_implemented_by
- part_of_directly, part_of

And one datatype properties:

- hasNumericalValue

For reasoning purposes GBO supports classification with:

- existential restrictions
- universal restrictions
- cardinality restrictions
- domain / range
- covering axioms
- transitive object properties with sub-properties
- inverse and symmetric properties
- disjoint and equivalent classes

In the following figures ([A.2-A.4](#)) a unified view connecting BFO, GBO, SOSA, OM-2 is shown with a follow up of all definitions in GBO.

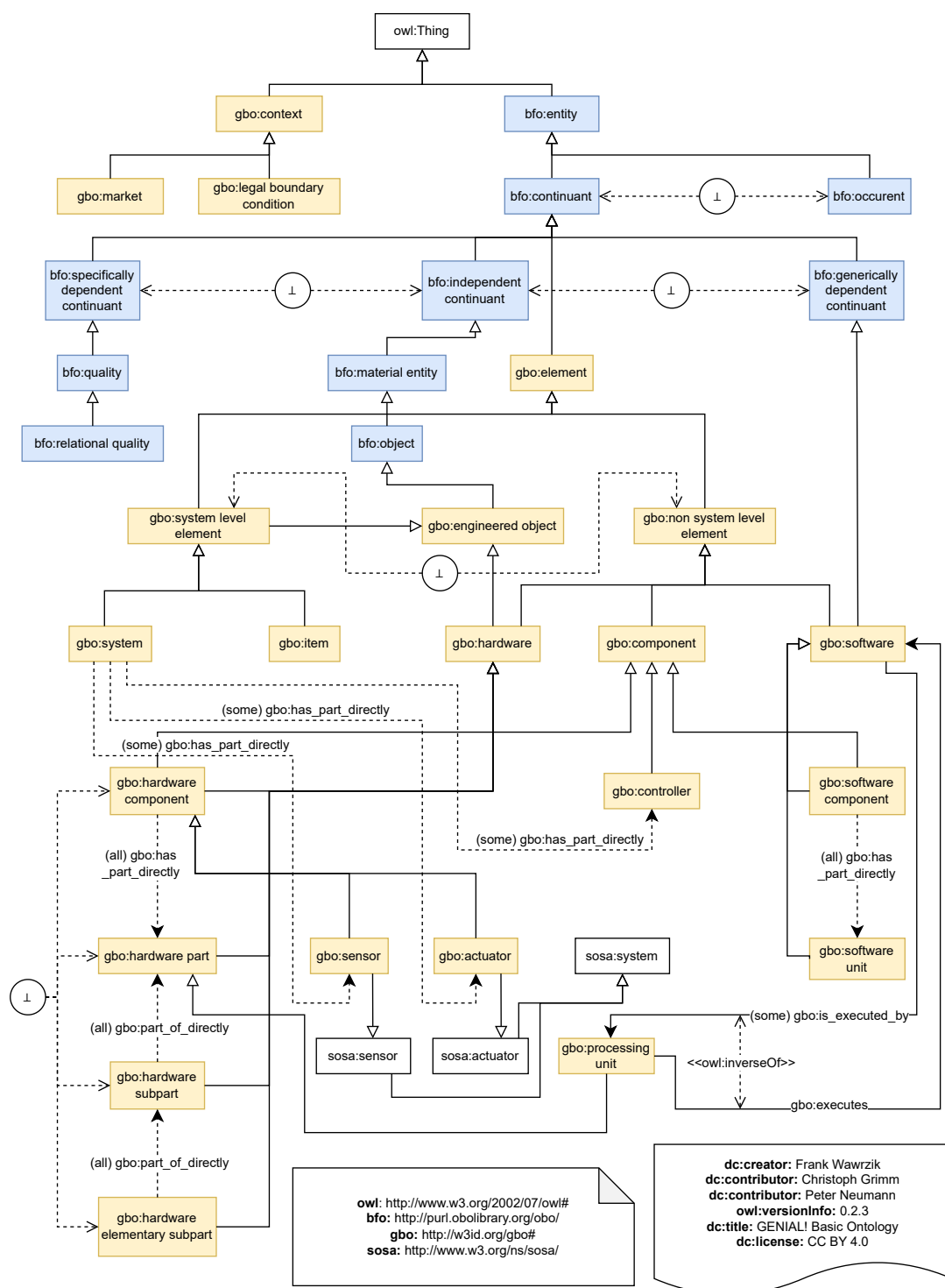


Figure A.2: GBO Core Part 1 Overview - Parts

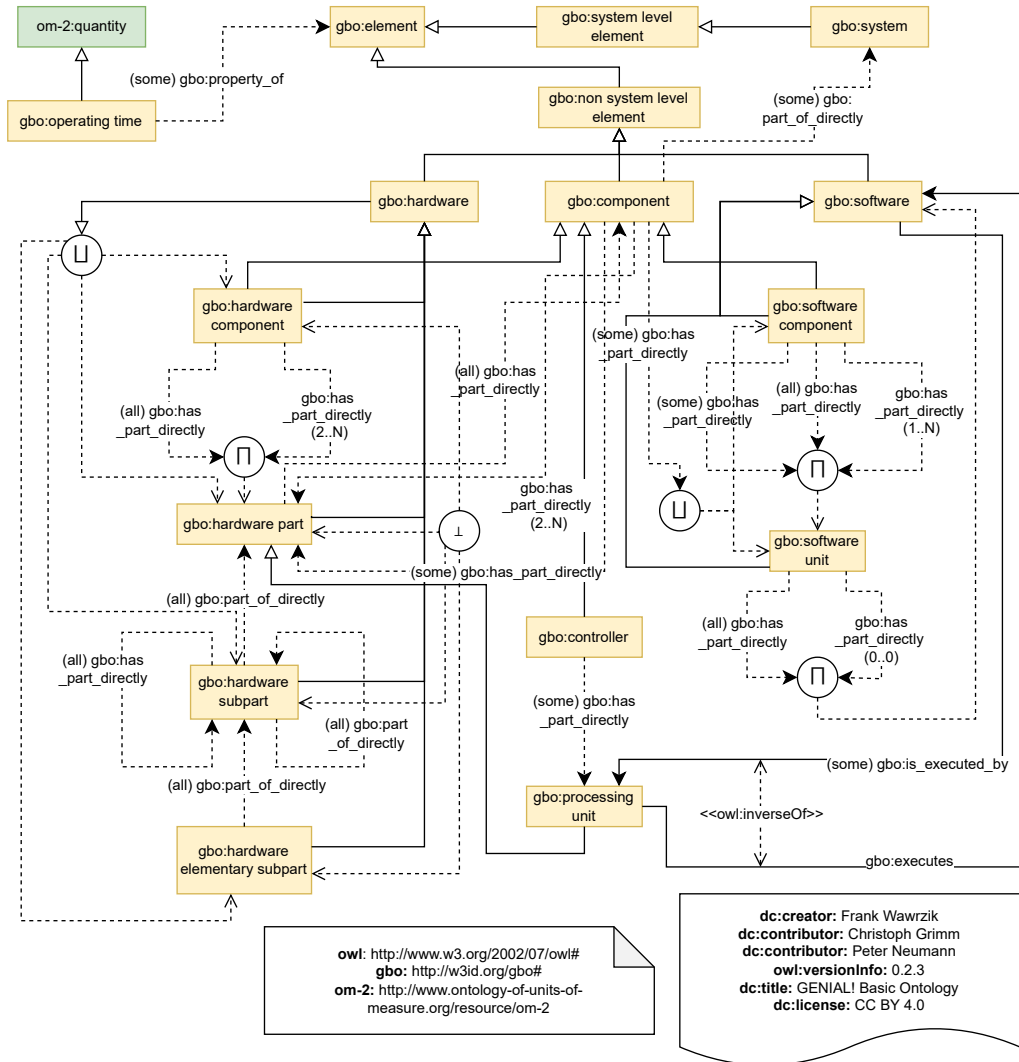


Figure A.3: More Detailed View on GBO Parts and Components

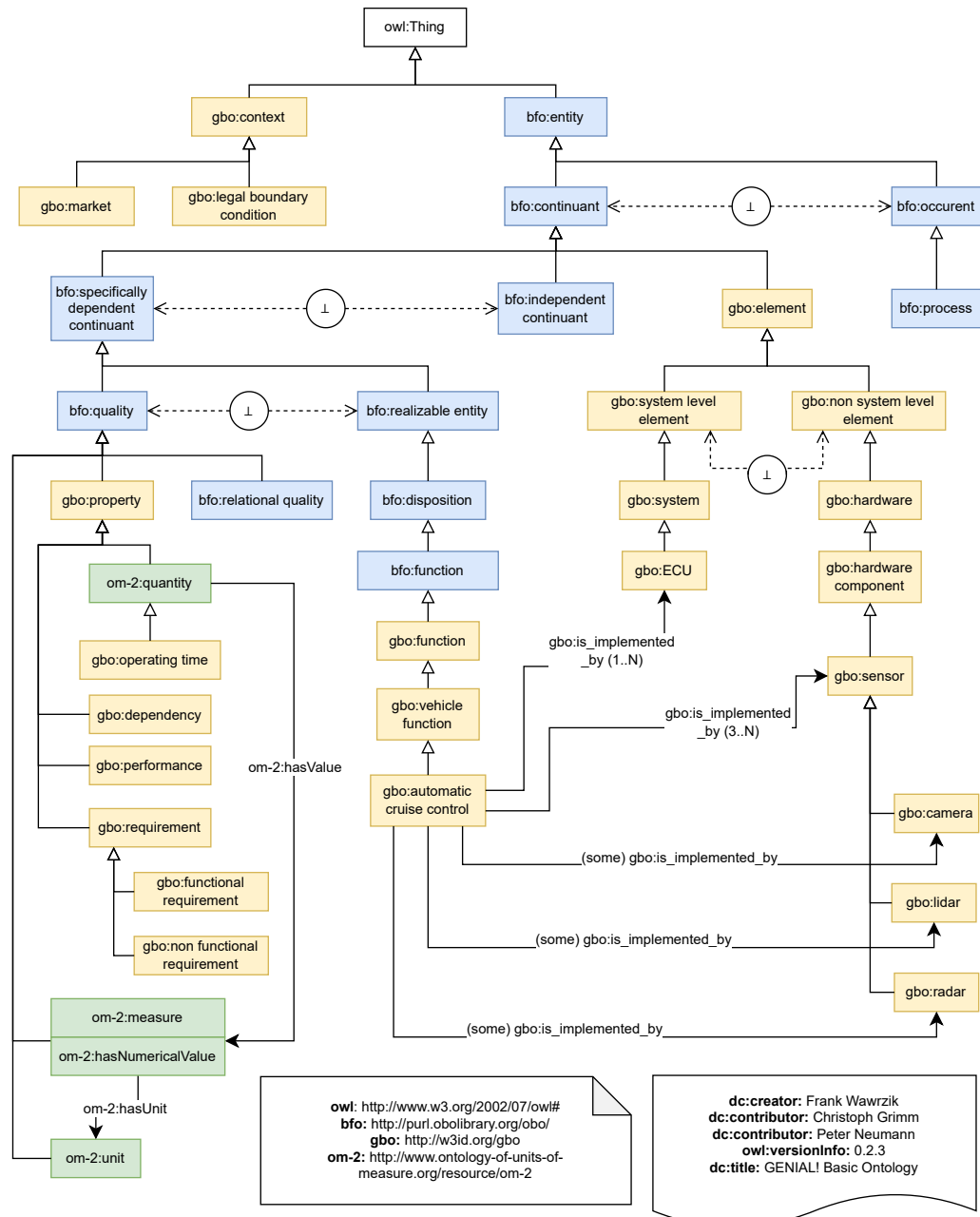


Figure A.4: GBO Core Part 2 Overview - Properties and Functions

Class	Definition
context	The circumstances that form the setting for an event, statement or idea and in terms of which it can be fully understood, e.g. Market, legal frame conditions
legal boundary condition	No definition in GBO 0.2.3
market	No definition in GBO 0.2.3
social object	An object or object aggregate that stands in a social role.
material object	No definition in GBO 0.2.3
engineered object	An object created, designed or engineered by humans.
mechanical object	An engineered object that has mechanical characteristics. They can be either dominant mechanical characteristics like in a vehicle or wheel or less dominant like in a battery.
element	Continuant that is (according to ISO 26262) a system (3.163), components (3.21) (hardware or software), hardware parts (3.71), or software units (3.159)
non system level element	An element that is not on the system-level
system level element	An element that is on the system level
item	System level element that is (according to ISO26262) system (3.163) or combination of systems (3.163), to which ISO 26262 is applied, that implements a function or part of a function at the vehicle level. Note 1 to entry: See vehicle function (3.178).
system	System level element that is according to ISO 26262: set of components (3.21) or subsystems that relates at least a sensor, a controller and an actuator with one another. Note 1 to entry: The related sensor or actuator can be included in the system, or can be external to the system.
electronic control unit / ECU	No definition in GBO 0.2.3
component	According to ISO26262: non-system level element (3.41) that is logically or technically separable and is comprised of more than one hardware part (3.71) or one or more software units (3.159). A component is a part of a system (3.163).
hardware component	No ISO definition. Formal Definition derived from other definitions.
software component	one or more softwareunits (3.159)
actuator	An actuator is a hardware component that implements some actuating function and is responsible for moving or controlling a mechanism or system. Not defined in ISO26262.

Table A.2: Classes and its Definitions in GBO V0.2.3 - Part 1

Class	Definition
sensor	Hardware component that implements some sensing function. Not defined in ISO26262.
controller	No definition in GBO 0.2.3
hardware subpart	portion of a hardware part (3.71) that can be logically divided and represents second or greater level of hierarchical decomposition
hardware elementary subpart	smallest portion of a hardware subpart (3.73) considered in safety (3.132) analysis
processing element / processing unit / processor	(according to ISO26262) hardware part (3.71) providing a set of functions for data processing, normally consisting of a register set, an execution unit, and a control unit
hardware part	a piece of hardware that is (according to ISO 26262) a portion of a hardware component (3.21) at the first level of hierarchical decomposition
hardware / hardware element	From definition of element: Note 1 to entry: When “software element” or “hardware element” is used, this phrase denotes an element of software only or an element of hardware only, respectively.
software / software element	From definition of element: Note 1 to entry: When “software element” or “hardware element” is used, this phrase denotes an element of software only or an element of hardware only, respectively.
embedded software	a software that is (according to ISO 26262) fully-integrated software to be executed on a processing element (3.113)
software unit	a piece of software that is (according to ISO26262) an atomic level software component (3.157) of the software architecture (3.1) that can be subjected to stand-alone testing (3.169)
measure	A bfo:quality that are amounts of quantities.
property	A quality or characteristic of an element, hardware, software or function. A property has a boolean, integer, or real variable through the measure class. Specifically properties can be unquantified and quantified. But quantified properties are quantities.

Table A.3: Classes and its Definitions in GBO V0.2.3 - Part 2

Class	Definition
dependency	A gbo:property that is a boolean or arithmetic function / equation on properties. Used to be calculated with GENIAL constraint propagation. As string in LaTeX syntax.
performance	a gbo:property on how well a person, machine, etc. does a piece of work or an activity
quantity	A quantity is a (property that is quantifiable and a) representation of a quantifiable (standardised) aspect (such as length, mass, and time) of a phenomenon (e.g., a star, a molecule, or a food product). Quantities are classified according to similarity in their (implicit) metrological aspect, e.g. the length of my table and the length of my chair are both classified as length.
operating time	A quantity that is (according to ISO 26262) a cumulative time that an item (3.84) or element (3.41) is functioning, including degraded modes.
requirement	A property that is wanted or needed.
functional requirement	A requirement that specifies something the system should do.
non functional requirement	A requirement that describes how the system works.
unit	A quality that is any standard used for comparison in measurements.
function	A bfo:function that an element (e.g. system, component, hardware or software) implements.
actuating	A gbo:function to change or control an object
processing	A gbo:function to calculate a set of instructions or software
sensing	A gbo:function to perceive an object
vehicle function	A gbo:function that is (according to ISO26262) behaviour of the vehicle, intended by the implementation of one or more items (3.84), that is observable by the customer.
automatic cruise control	An “automatic cruise control” is a vehicle function that can be implemented, using different ECUs and a variety of sensor technology (e.g. Radar, Lidar, Camera).

Table A.4: Classes and its Definitions in GBO V0.2.3 - Part 3

The purpose is just to show a few components and application examples. The basis of the domain (classes) were a neural net accelerator use case, wikipedia and a simulation library of cyber physical systems. The knowledge base consists of the instantiation of the domain with concrete parts and values of the properties, which already contain a lot of data.

[illegible]

Figure A.5: TBox Reasoning of Digital Filter System

The example shown in figure A.5 demonstrates TBox classification of the domain with GBO. Strict universal restriction require hierarchical conformance as can be seen with the axioms of component. To be a component two hardware parts and software is needed. A link to a different hierarchical class yields an error. Only in this way consistency with the standard is ensured. Which was one of the purposes. The filter class has the functions signal processing and transfer function, which are inherited by all filters and also by the digital filter system. The digital filter system passes the reasoner test. The axioms and related other parts help determining its position in the ontology and is a basis for further classifications. As can be seen a "system" is in this case a component, which is not a contradiction as we work with namespaces in the semantic web.

Sensor Types Taxonomy

Figure A.6 is classified according to domain experts of the creators of the ZVEI roadmap (<https://www.zvei.org/presse-medien/publikationen/technologie-roadmap-next-generation/>).

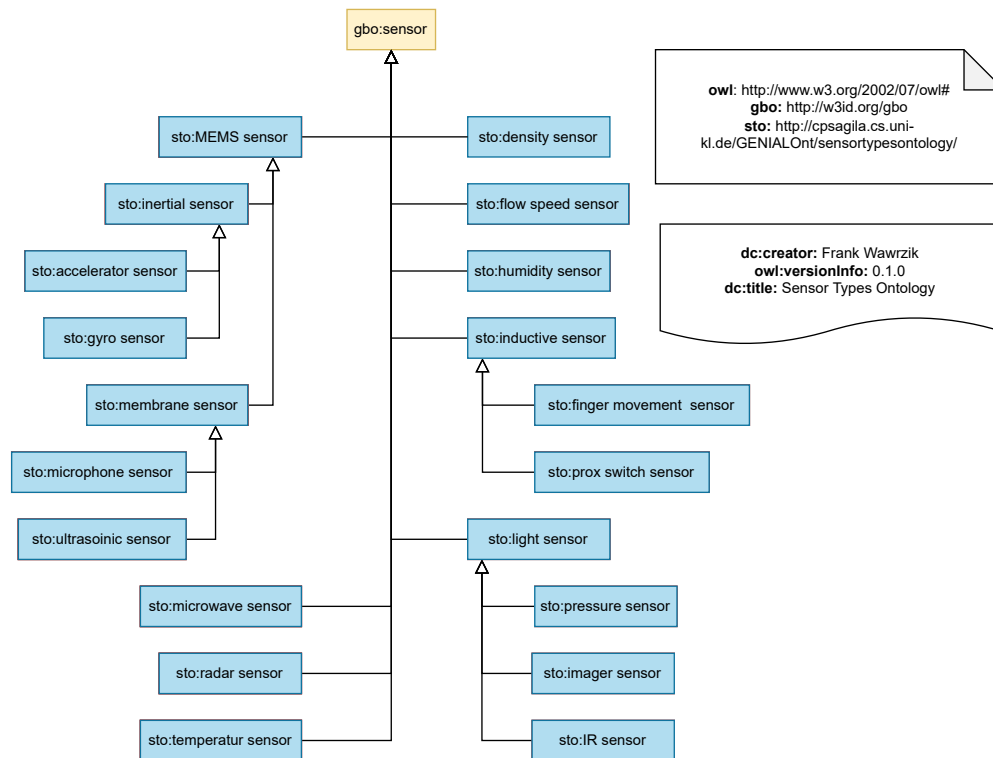


Figure A.6: Basic Sensor Types Taxonomy

Class	Definition
integrated circuit	An integrated circuit or monolithic integrated circuit (also referred to as an IC, a chip, or a microchip) is a set of electronic circuits on one small flat piece (or "chip") of semiconductor material, usually silicon.
flash array	Flash storage is a data storage technology based on high-speed, electrically programmable memory. The speed of flash storage is how got its name: It writes data and performs random I/O operations in a flash.
all flash array / afa	An all-flash array (AFA) is a storage infrastructure that contains only flash memory drives instead of spinning-disk drives. All-flash storage is also referred to as a Solid-State Array (SSA). AFAs and SSAs offer speed, performance and agility for your business applications.
dynamic random access memory	(pronounced DEE-RAM), is widely used as a computer's main memory. Each DRAM memory cell is made up of a transistor and a capacitor within an integrated circuit, and a data bit is stored in the capacitor. Since transistors always leak a small amount, the capacitors will slowly discharge, causing information stored in it to drain; hence, DRAM has to be refreshed (given a new electronic charge) every few milliseconds to retain data.
instruction set	The complete set of all the instructions in machine code that can be recognized and executed by a central processing unit. 'A Script is an instruction set used to execute operations on a controller device.'
mfcc computation	MFCC (Mel Frequency Cepstral Co-efficients) is a kind of feature in audio processing. They are used as an input for a CNN for wakeword/keyword detection. The coefficients aim to represent the phonemes the human voice and language use.
temperature	The degree or intensity of heat present in a substance or object, especially as expressed according to a comparative scale and shown by a thermometer or perceived by touch.
voltage	Voltage is the pressure from an electrical circuit's power source that pushes charged electrons (current) through a conducting loop, enabling them to do work such as illuminating a light.
execute code	Function which executes some piece of code or software.

Table A.5: A Selection of Classes and its Definitions in HW / SW Domain Ontology V0.1.1

A.1.5 Complete APPEL Model for Constraint Exploration

```

Appel(agilaModel){
  +"Package FrankSimple"
  +" // Car Model *****"
  +" Note: the property value can stand for any kind "
  +" of property, material or immaterial and is exemplary"
  +" Context isA Any"
  +" Time isA Context"
  +" time1 isInstanceOf Time"
  +"   it hasProperty tinminutes : Real (0.0..520000.0)"
  +"   it hasProperty tinyears: Real (5.0..5.0)"
  +" Wheel isA Element"
  +"   it hasProperty value : Real (60.0..60.0)"
  +" Body isA Element"
  +"   it hasProperty value : Real (2000.0..2000.0)"
  +" ChassisPart1 isA Element"
  +"   it hasProperty value : Real (1500.0..1500.0)"
  +" ChassisPart2 isA Element"
  +"   it hasProperty value : Real (1500.0..1500.0)"
  +" Chassis isA Element"
  +"   it hasA chassispartp1 : Int (1..1) ChassisPart1"
  +"   it hasA chassispartp2 : Int (1..1) ChassisPart2"
  +"   it hasProperty value : Real = SUM(value)"
  +" Powertrain isA Element"
  +" Engine isA Element"
  +" ElectricEngine isA Engine"
  +"   it hasProperty value : Real (4000.0..4000.0)"
  +" CombustionEngine isA Engine"
  +" Battery isA Element"
  +" it dependsOn FrankSimple.time1"
  +" it dependsOn FrankSimple.Inflation"
  +"   it hasProperty startvalue : Real (14000.0..14000.0) "
  +"   it hasProperty value : Real = startvalue*(powerb(0.94408, tinyears))"
  stmt = "   it hasProperty batteryvaluepredictionout : Real = value"
  println("batteryvaluepredictionout = " + getVar("batteryvaluepredictionout").
    aadd().getRange() + " ")
  +"   it hasProperty batteryvaluepredictionwithinflation : Real = value"

  +" EVPowertrain isA Powertrain"
  +"   it hasA electricengine1 : Int (1..1) ElectricEngine"
  +"   it hasA battery1 : Int (1..1) Battery"
  +"   it hasProperty value : Real = SUM(value)"
  +" ICEPowertrain isA Powertrain"
  +"   it hasProperty value : Real (4000.0..4000.0)"

  //" city1 isInstanceOf City"
  //" Property population : Real (40000.0..40000.0)"
  +" Car isA Element"
  +" ElectricCar isA Car"
  //" it hasA wheel : Int (4..4) Wheel"
  +"   it hasA body1 : Int (1..1) Body"
  +"   it hasA chassisp1 : Int (1..1) Chassis"
  +"   it hasA chassisp2 : Int (1..1) Chassis"

```

```

//+" it hasA battery : Int (1..1) Battery"
+" it hasA evpowertrain1 : Int (1..1) EVPowertrain"
+" it hasProperty ElectricalCarvalue : Real = SUM(value)"
stmt = "          it hasProperty ElectricalCarvalueout : Real = ElectricalCarvalue"
println("ElectricalCarvalueout = " + getVar("ElectricalCarvalueout").
      aadd().getRange() + " ")

+" PassengerCar isA Car"
+" it hasProperty PassengerCarvalue : Real (22000.0..22000.0)"
stmt = "          it hasProperty PassengerCarvalueout : Real = PassengerCarvalue"
println("PassengerCarvalueout = " + getVar("PassengerCarvalueout").
      aadd().getRange() + " ")
+" Car isA Element"
+" PassengerCar isA Car"
+" it hasA wheel : Int (1..1) Wheel"
+" it hasA body : Int (1..1) Body"
+" it hasA chassis : Int (1..1) Chassis"
+" it hasProperty Carvalue : Real = SUM(value)"
stmt = "          it hasProperty Carvalueout : Real = Carvalue"
println("Carvalueout = " + getVar("Carvalueout").aadd().getRange() + " ")

+" // Some Tests *****"
//+" Wheel hasA Car : Cartest"
//+" Car hasElement Body"
//+" Car hasElement Chassis"
//+" Inflation isA Context"
//+" it dependsOn FrankSimple.Wheel"
//+" it dependsOn FrankSimple.Body"
//+" it dependsOn FrankSimple.Chassis"
//+" it dependsOn FrankSimple.time1"
//+"      //Property inflinyears : Real = Body.value*(1.0-0.02*tinyyears)"
//+"      Property Carvalue : Real = SUM(Car.parts.value)"
//+"      //Property infl : Real = value*(1.0-0.02*tinminutes/(60.0*24.0*360.0))"
//stmt = "          Property D : Real = Carvalue"
//println("D = " + getVar("D").aadd().getRange() + " ")
//+" C isA Component"
//+" it dependsOn FrankSimple.city1"
//+" it hasProperty K : Real = population"
//stmt = "          Property J : Real = K"
//println("J = " + getVar("J").aadd().getRange() + " ")
//+"      Property D : Real (3.0..5.0)"
//+"      Property F : Real (10.0..20.0)"
//+"      Property Z : Real"
//+" Flying isA Element"
//+"      it dependsOn Test"
//+"      it dependsOn Wheel"
//+" Flying dependsOn Processor.K"
//+" Flying dependsOn Wheel.wheelspeed"
//stmt = "          Property J : Real = K * wheelspeed"
//println("J = " + getVar("J").aadd().getRange() + " ")
//stmt = "          Property Z : Int = K * J + 2*F + X"
//stmt = "          Property X : Int = F * Z"

```

```

+//Ultratrail Constraints *****"
+DeepNeuralNetworkLayer isA Software"
+it hasProperty C: Real (32.0..32.0)//num. input channel"
+it hasProperty C_w: Real (25.0 .. 65.0) //width of input channel"
+it hasProperty K: Real (48.0 .. 48.0) // num. of output channels"
+it hasProperty F: Real (27.0 .. 27.0) // filter width"
+it hasProperty ns: Real (2.0 .. 2.0) // stride"
//+Property p: Bool //padding"

+UltraTrail isA Processor"
+ it executes DeepNeuralNetworkLayer"
+ it dependsOn FrankSimple.DeepNeuralNetworkLayer"
+ it dependsOn FrankSimple.MooresLaw"
defVar("i", scalar(0.0))
defVar("p", True)
//defVar("C", range(32.0 .. 34.0))
+it hasProperty C: Real (32.0..32.0)//num. input channel"
+ it hasProperty fclk: Real (0..1e9) [Hz]"
stmt = "it hasProperty s: Real = ns"
println("s = [" + getVar("s").aadd().getRange().min + ", " +
    getVar("s").aadd().getRange().max + "]")

//+ Property s: Real = power2(executes.DeepNeuralNetworkLayer.ns)"
//stmt = " Property sout : Real = s"
//println("sout = [" + getVar("sout").aadd().getRange().min + ", " +
    getVar("sout").aadd().getRange().max + "]")
//+ Property C_w_hat: Real = ITE(p, C_w + F, C_w)"
//+ Property C_w_hat: Real = ITE(p, C_w + F, C_w)"
//stmt = "Property C_w_hatout: Real = C_w_hat"
stmt = "it hasProperty C_w_hat: Real = ITE(p, C_w + F, C_w)" // eq. 8
println("C_w_hat = [" + getVar("C_w_hat").aadd().getRange().min + ", " +
    getVar("C_w_hat").aadd().getRange().max + "]")
//+ Property C_w_hat: Real = ITE(p, C_w + F, C_w)"
//+ Property C_wb: Real = F / 2.0"
stmt = "it hasProperty a_w: Real = ((C_w_hat - F) / s + 1.0)" // eq. 9
println("a_w = [" + getVar("a_w").aadd().getRange().min + ", " +
    getVar("a_w").aadd().getRange().max + "]")

stmt = "it hasProperty C_wb: Real = F / 2.0" // eq. 10
println("C_wb = [" + getVar("C_wb").aadd().getRange().min + ", " +
    getVar("C_wb").aadd().getRange().max + "]")
//assertEquals(4.5, getVar("C_wb").aadd().getRange().min, 0.00001)
//assertEquals(4.5, getVar("C_wb").aadd().getRange().max, 0.00001)

stmt = "it hasProperty a_pb: Real = ITE(p, (C_wb - 1.0) / s + 1.0, 0.0)"
// eq. 11
println("a_pb = [" + getVar("a_pb").aadd().getRange().min + ", " +
    getVar("a_pb").aadd().getRange().max + "]")
//assertEquals(2.75, getVar("a_pb").aadd().getRange().min, 0.00001)
//assertEquals(2.75, getVar("a_pb").aadd().getRange().max, 0.00001)

```

```

// Sum 0 ... 2.65-1 over 7.5 - 4i (0, 1, 2, 3?) = 7.5*3-4-8 (-12?)
stmt = "it hasProperty MAC_notb: Real = sum_i(0.0, a_pb - 1.0, F / 2.0 - s * i )"
// eq. 12
println("MAC_notb = [" + getVar("MAC_notb").aadd().getRange().min + ", " +
    getVar("MAC_notb").aadd().getRange().max + "]")
//assertEquals(7.0, getVar("MAC_notb").aadd().getRange().min, 0.00001)
//assertEquals(7.0, getVar("MAC_notb").aadd().getRange().max, 0.00001)

//stmt = "Property Fw: Real = a_w * s + F - s" //eq. 13
stmt = "it hasProperty Fw: Real = a_w * s + F - s" //eq. 13
println("Fw = [" + getVar("Fw").aadd().getRange().min + ", " +
    getVar("Fw").aadd().getRange().max + "]")
//assertEquals(34.0, getVar("Fw").aadd().getRange().min, 0.00001)
//assertEquals(34.0, getVar("Fw").aadd().getRange().max, 0.00001)

stmt = "it hasProperty C_we: Real = Fw - C_w - C_wb" // eq. 14
println("C_we = [" + getVar("C_we").aadd().getRange().min + ", " +
    getVar("C_we").aadd().getRange().max + "]")
//assertEquals(4.5, getVar("C_we").aadd().getRange().min, 0.00001)
//assertEquals(4.5, getVar("C_we").aadd().getRange().max, 0.00001)

stmt = "it hasProperty a_pe: Real = ITE(p, (C_we - 1.0) / s + 1.0, 0.0)"
// eq. 15
println("a_pe = [" + getVar("a_pe").aadd().getRange().min + ", " +
    getVar("a_pe").aadd().getRange().max + "]")
//assertEquals(2.75, getVar("a_pe").aadd().getRange().min, 0.00001)
//assertEquals(2.75, getVar("a_pe").aadd().getRange().max, 0.00001)

stmt = "it hasProperty MAC_note: Real = sum_i(0.0, a_pe - 1.0, F / 2.0
    - s * i - (C_wb - C_we))" // eq. 16
println("MAC_note = [" + getVar("MAC_note").aadd().getRange().min + ", " +
    getVar("MAC_note").aadd().getRange().max + "]")
//assertEquals(7.0, getVar("MAC_note").aadd().getRange().min, 0.00001)
//assertEquals(7.0, getVar("MAC_note").aadd().getRange().max, 0.00001)

stmt = "it hasProperty t_l: Real = 1.0 + C / 8.0 K / 8.0
(a_w * F - MAC_notb - MAC_note)" // eq. 17
println("t_l = [" + getVar("Fw").aadd().getRange().min + ", " +
    getVar("t_l").aadd().getRange().max + "] clock cycles")

CspSolver(agilaModel).solve("t_l", 1)
var result : AADD = CspSolver(agilaModel).
    calcPropValFromSerializedChangesOfIndependentVars(this.symbolTable.builder,
        symbolTable.getVar("t_l"), symbolTable.getVar("C"), symbolTable.builder
        .range(33.0, 33.0, "C"))
) as AADD
//println("C = [" + getVar("C").aadd().getRange().min + ", " +
    getVar("C").aadd().getRange().max + "]")
println("t_l = [" + getVar("t_l").aadd().getRange().min + ", " +
    getVar("t_l").aadd().getRange().max + "] clock cycles")
//assertEquals(2581.0, getVar("t_l").aadd().getRange().min, 0.00001)
//assertEquals(2581.0, getVar("t_l").aadd().getRange().max, 0.00001)

```

```

//assert(getVar("t_1").aadd().getRange().contains(2521.0))
println("result = [" + result.getRange().min + ", " + result.getRange().max + "]")
//println("t_1 = [" + getVar("t_1").aadd().getRange().min + ", " +
    getVar("t_1").aadd().getRange().max + "] clock cycles")
//println("C = [" + getVar("C").aadd().getRange().min + ", " +
    getVar("C").aadd().getRange().max + "]")

+//Roadmap Model *****"
+" Inflation isA Context"
+" it dependsOn FrankSimple.ElectricCar"
+" it dependsOn FrankSimple.time1"
+" it dependsOn FrankSimple.UltraTrail"
+" it dependsOn FrankSimple.Battery"

//Inflation in 2020 is 2%
//else{
+" //inflation for battery value prediction"
+" it hasProperty inflationbatteryvalueprediction: Real =
    value*powerb(0.98,tinyyears)"
stmt = " it hasProperty inflationbatteryvaluepredictionout : Real =
    inflationbatteryvalueprediction"
println(
    "inflationbatteryvaluepredictionout = " +
        getVar("inflationbatteryvaluepredictionout").aadd().getRange() + " "
)

//+" Property inflation: Real = Carvalue-(Carvalue*0.02*tinyyears)"
//+" it hasProperty inflationbattery: Real =
    ElectricalCarvalue*power2(0.02*tinyyears)"
+" it hasProperty inflation: Real = ElectricalCarvalue*powerb(0.98,tinyyears)"
stmt = " it hasProperty inflationout : Real = inflation"
println("inflationout = " + getVar("inflationout").aadd().getRange() + " ")
//}

+" MooresLaw isA Context"
+" it dependsOn FrankSimple.UltraTrail"
+" it dependsOn FrankSimple.time1"
+" it dependsOn FrankSimple.DeepNeuralNetworkLayer"
//+" it dependsOn FrankSimple.Processor"
//+" Property X : Real = ns"
+" it hasProperty runtimein : Real = t_1"
//works only for t>1
//+" it hasProperty runtimeout : Real = runtimein/power2(0.5*tinyyears)"
+" it hasProperty runtimeout : Real = runtimein/power2(0.5*tinyyears)"
stmt = " it hasProperty runtimeout2 : Real = runtimeout"
println("runtimeout2 = " + getVar("runtimeout2").aadd().getRange() + " ")
//println("C_w = [" + getVar("DeepNeuralNetworkLayer.C_w").
    aadd().getRange().min + ", " + getVar("DeepNeuralNetworkLayer.C_w").
    aadd().getRange().max + "]")
}

```

Note: Some terms here are not yet completely aligned with the ontology and for that purpose might be readjusted in the future. This was because the metamodel in the backend was also frequently changing.

A.1.6 AGILA ArangoDB Knowledge Base

This subsection gives a short overview of the database content. Reasons for choosing a database were speed, volume and data access among others. Figure A.7 shows a pulpissimo controller instance of component and figure A.8 some more details with software.

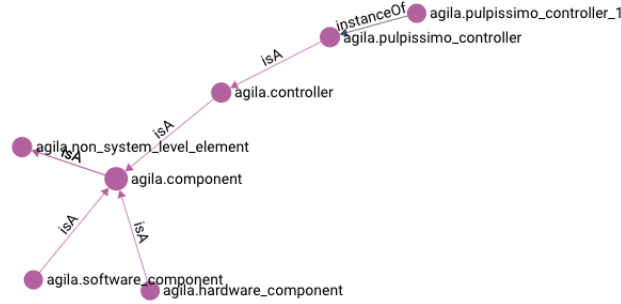


Figure A.7: An Instance of a Pulpissimo Controller as GBO Component



Figure A.8: Some Software of the Pulpissimo Controller

Figure A.9 shows the cnn accelerator of the controller with parts, properties and functions as a bended graph. Figure A.10 also shows more parts and properties.

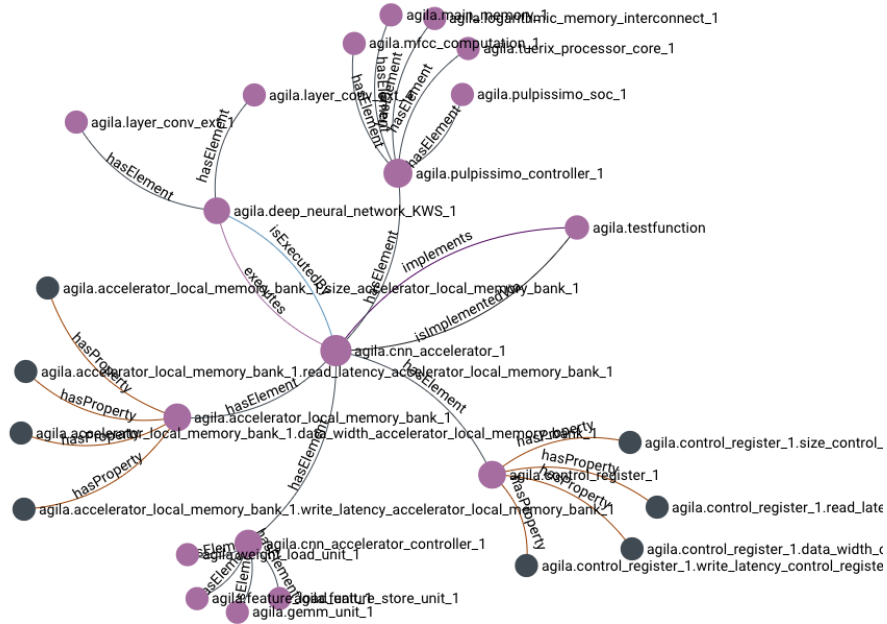


Figure A.9: Bended Graph with Inverse Relations

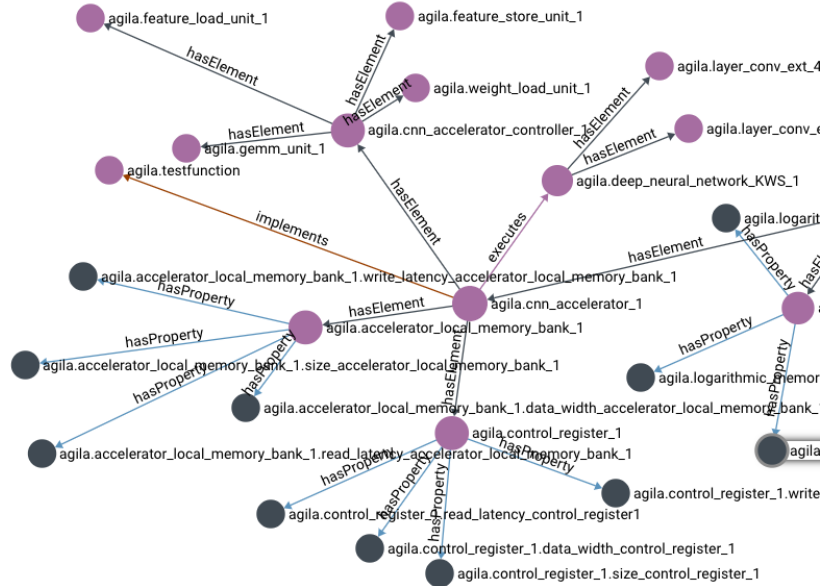


Figure A.10: Graph with More Nodes

```

// OntologyService.java
package com.github.tukcps.agila.backend.ontology;

import com.github.tukcps.agila.backend.entityRequestModel.
    ElementDetailsRequestModel;
import com.github.tukcps.agila.backend.entityRequestModel.
    InstanceDetailsRequestModel;
import com.github.tukcps.agila.backend.repository.ElementRepository;
[...]

/**
 * @author Frank Wawrzik
 * @date June 2021
 * This class translates basic constructs of OWL into the ArangoDB database
 * as a REST-Service
 */
@Service
public class OntologyService {

    @Autowired
    private ElementService elementService;
    @Autowired
    private ElementRepository elementRepository;
    [...]

    public OntologyService() {}
    public void OWLTranslation() throws CloneNotSupportedException {
    [...]
    try {
        ontologyFile = new File("/Users/Frank/local/git/GENIALOntologies/
        OntologyModuleSuite/GENIAL!BasicOntology/GENIALOntBFO.owl");
        ontologyFiledomainhardware = new File("/Users/Frank/local/git/
        GENIALOntologies/OntologyModuleSuite/CarModelOntologies/domainhardware.owl");
        ontologyFiledomainhardwareKB = new File("/Users/Frank/local/git/
        GENIALOntologies/OntologyModuleSuite/CarModelOntologies/domainhardwareKB.owl");
        owlOntology = owlOntologyManager.loadOntologyFromOntologyDocument
            (ontologyFile);
        owlOntologydomainhardware = owlOntologyManager.
            loadOntologyFromOntologyDocument(ontologyFiledomainhardware);
        [...]

        //Initialize reasoner
        OWLDataFactory df = OWLManager.getOWLDataFactory();
        StructuralReasonerFactory srf = new StructuralReasonerFactory();
        OWLReasoner reasoner = srf.createReasoner(owlOntology1);
        ElementDetailsRequestModel elementDetailsRequestModel = new
            ElementDetailsRequestModel();

        //Get all classes from the ontology
        Set<OWLClass> myClasses = owlOntology1.getClassesInSignature(true);
        [...]
    }
}

```

Figure A.12: Ontology Service Loading Ontologies and Initializing Reasoner

```

[...]
```

```

// Write the triples
Set<OWLNamedIndividual> myIndividuals =
    owlOntology1.getIndividualsInSignature();

for (OWLNamedIndividual myIndividual: myIndividuals){
    System.out.println("Individuals in the ontology: " +
        myIndividual.getIRI().getShortForm());
    //write each individual in a collection
    Optional<Element> instanceElement =
        elementRepository.findElementById("agila" + "." +
            myIndividual.getIRI().getShortForm());
    if (instanceElement.isEmpty()) {
        elementDetailsRequestModel.setName(myIndividual.getIRI().
            getShortForm());
        Element element = new Element("agila" + "." +
            + elementDetailsRequestModel.getName());
        element.isInstance = true;
        Element returnedElement = elementService.createElement(element);
    }

    Set<OWLObjectPropertyAssertionAxiom> myOWLPropertyAssertionAxioms =
        owlOntology1.getObjectPropertyAssertionAxioms(myIndividual);

    for (OWLObjectPropertyAssertionAxiom myOWLPropertyAssertionAxiom:
        myOWLPropertyAssertionAxioms){
        [...]
```

```

//write object of the triple first
Optional<Element> objectElement =
    elementRepository.findElementById("agila" + "." +
        myOWLPropertyAssertionAxiom.getObject().asOWLNamedIndividual().
            getIRI().getShortForm());
    if (objectElement.isEmpty()) {
        elementDetailsRequestModel.setName(myOWLPropertyAssertionAxiom.
            getObject().asOWLNamedIndividual().getIRI().getShortForm());
        Element element = new Element("agila" + "." +
            elementDetailsRequestModel.getName());
        element.isInstance = true;
        Element returnedElement = elementService.createElement(element);
    }
    //write "executes" object property
    if (myOWLPropertyAssertionAxiom.getProperty().
        asOWLObjectProperty().getIRI().getShortForm().equals("executes")){
        elementService.setExecutes("agila."+myIndividual.getIRI().
            getShortForm(), "agila."+ myOWLPropertyAssertionAxiom.getObject().
            asOWLNamedIndividual().getIRI().getShortForm());
    }
}
[...]
```

```

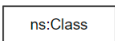
}
```

Figure A.13: Ontology Service Writing ABox Triples to Database (Excerpt)

A.2 Ontology Notation and Legend

The ontology overview in this appendix are created with diagrams.net and the chalk notation ¹. In order to independently understand the diagrams, all needed visual OWL constructs are listed in the following for reference.

A.2.1 Basic Elements

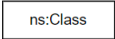
Diagram Block	Description	OWL Element
	Block to represent named and unnamed classes, as well as individual elements within the ontology conceptualization. The content of the block should be accompanied with the prefix and the name of the concept on order to fully identify it.	<code>owl:Class</code>
	Block to represent named and unnamed classes, as well as individual elements within the ontology conceptualization. The content of the block should be accompanied with the prefix and the name of the concept on order to fully identify it.	<code>owl:Individual</code>
	Standard way to represent object properties. Variations can apply to the type of line or the connections style depending on the range or domain specification. For more details see section 2.3.	<code>owl:ObjectProperty</code>
	Special arrow to indicate sub-class relationship between two classes.	<code>owl:subClassOf</code>
	Special arrow to represent several relationships between elements of this specification. It can be used to indicate <code>rdf:type</code> relationships, or to connect a <code>owl:unionOf</code> axiom with all the concepts it is composed of.	<code>rdf:type</code>
	Standard way to represent datatype properties attached to a specific owl:Class element. Variations can apply to the type of outer line depending on the domain and range specification. For more details see section 2.4.	<code>rdf:DatatypeProperty</code>
	Block to indicate all the namespaces used in the ontology. The first namespace is the URI used for the current ontology. It is obligatory to include all the namespaces being used in order to use the ontology converter service.	<code>@prefix base:</code> <code><http://namespace.com#></code>
	Block to indicate the annotation properties describing the ontology. The annotations in use should include the prefix and the annotation name, as indicated in the figure. If custom annotations are utilized, the namespace block should include the prefixes and namespaces for those annotation properties.	<code>owl:AnnotationProperty</code>

¹<https://chowlk.linkeddata.es/notation.html>

A.2.2 Classes

Class Definition

Definition of a named class.

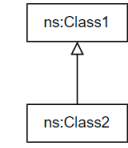
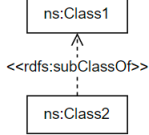
Diagram Block	OWL Element
	<code>owl:Class</code>

Definition of an unnamed class to represent logical combinations between other classes, such as AND or OR operators.

Diagram Block	OWL Element
	<code>owl:Class</code>

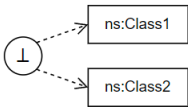
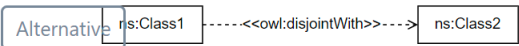
Sub-Class

Graphical representations to indicate that `ns:Class2` concept is sub-class of `ns:Class1`.


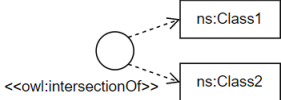
Diagram Block	OWL Element
<div>Preferred</div> 	<code>ns:Class2 owl:subClassOf ns:Class1</code>
<div>Alternative</div> 	

Disjoint Classes


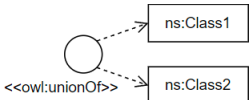
Graphical representations to indicate that `ns:Class2` and `ns:Class1` are disjoint concepts

Diagram Block	OWL Element
<div>Preferred</div> 	<code>ns:Class1 owl:disjointWith ns:Class2</code>
<div>Alternative</div> 	

Intersection of Classes

Diagram Block	OWL Element
<div>Preferred</div> 	<code>owl:intersectionOf (ns:Class1 ns:Class2)</code>
<div>Alternative</div> 	



Union of Classes

Diagram Block	OWL Element
<div>Preferred</div> 	<code>owl:unionOf (ns:Class1 ns:Class2)</code>
<div>Alternative</div> 	

A.2.3 Object Properties

Domain and Range

Object properties without domain and range.

Diagram Block	OWL Element
<div>Preferred</div> 	<code>ns:objectProperty rdf:type owl:ObjectProperty .</code>
<div>Alternative</div> 	

Object properties with domain and range.

Diagram Block	OWL Element
	<pre>ns:objectProperty rdf:type owl:ObjectProperty ; rdfs:domain ns:Class1 ; rdfs:range ns:Class2 .</pre>
	

Universal Restrictions

Universal restriction between 2 concepts. Concept **ns:Class1** is sub-class of an anonymous concept which has an object property **ns:objectProperty**, where all the individuals for this property should be of type **ns:Class2**.

Diagram Block	OWL Element
	<pre>ns:Class1 rdf:type owl:Class ; rdfs:subClassOf [rdf:type owl:Restriction ; owl:onProperty ns:objectProperty ; owl:allValuesFrom ns:Class2] .</pre>
	
	

Existential Restrictions

Universal restriction between 2 concepts. Concept **ns:Class1** is sub-class of an anonymous concept which has an object property **ns:objectProperty**, where all the individuals for this property should be of type **ns:Class2**.

Diagram Block	OWL Element
	<pre> ns:Class1 rdf:type owl:Class ; rdfs:subClassOf [rdf:type owl:Restriction ; owl:onProperty ns:objectProperty ; owl:someValuesFrom ns:Class2] . </pre>

Cardinality Restrictions

Cardinality restriction of a concept on an object property. The **ns:Class1** class is subclass of an anonymous concept which has an object property **ns:objectProperty**, and should have at least N1 and at most N2 individuals from class **ns:Class2**. If the N2 element is equal to the letter N, it means **owl:maxQualifiedCardinality** does not exist.

Diagram Block	OWL Element
	<pre> ns:Class1 rdf:type owl:Class ; rdfs:subClassOf [rdf:type owl:Restriction ; owl:onProperty ns:objectProperty ; owl:minQualifiedCardinality "N1"^^xsd:nonNegativeInteger ; owl:onClass ns:Class2] , [rdf:type owl:Restriction ; owl:onProperty ns:objectProperty ; owl:maxQualifiedCardinality "N2"^^xsd:nonNegativeInteger ; owl:onClass ns:Class2] . </pre>

A.2.4 Datatype Properties

Domain and Range

Datatype properties with domain and without range.

Diagram Block	OWL Element
	<pre> ns:datatypeProperty rdf:type owl:DatatypeProperty ; rdfs:domain ns:Class . </pre>

A.3 The SICYPHOS Framework and a SysML Design Methodology

From the paper "Modeling and Simulation with SICYPHOS" by the author [Waw+15]:
"The development of Cyber-Physical Systems is a challenge that involves a number of stakeholders from different disciplines that must co-operate and communicate. Customers, managers and developers and testers with expertise in different fields, such as mechanical, electrical or software systems are all involved in the development process. To increase accuracy of communication between stakeholders, informal text-based documents are being more and more replaced with models. However, different levels require different modeling languages. Figure A.14 gives an overview of the development process. At the system level, lightweight languages like SysML [Sta15] have become popular to describe the overall system requirements and structure. After specification and systems engineering, the components in different domains are developed using Domain-Specific Languages (DSL) such as C++ for software development, SystemC/SystemC-AMS for electronic systems, and Modelica for mechanical systems. To facilitate the seamless transition from SysML to DSL, tools such as Enterprise Architect provide code generation functionality. However, this functionality is oriented to single-domain code generation,

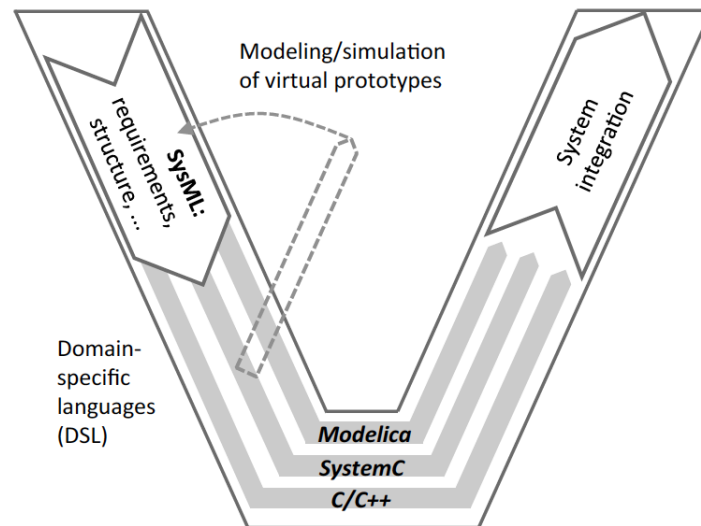


Figure A.14: SysML and domain-specific languages in the V diagram [Waw+15]

and it is insufficient to generate multi-domain models. Furthermore, it cannot benefit from reusability of preexisting Intellectual Property (IP) block implementations. Both different interacting domains and large IP components are essential in Cyber-Physical Systems development, and therefore, they require new approaches from research. In this paper, we give an overview of SICYPHOS (Simulation of CYber PHysical Systems). SICYPHOS is a simulation framework that integrates SysML, Modelica, SystemC, and

C/C++ in a seamless way.

Here, we give an overview of the SICYPHOS framework and its underlying concepts and objectives. We in particular describe concepts to achieve the intended capability of SICYPHOS to generate not only DSL, but also interfaces between different domains. These interfaces enable multidomain model generation, as well as interfaces between IP blocks and software components to be developed.

SICYPHOS. SICYPHOS is a SystemC-based framework for **Simulation of Cyber-Physical Systems**. The SICYPHOS Framework aims to assist at demonstrating a novel design methodology for Cyber-Physical Systems. Figure A.15 gives an overview of the overall framework and its design methodology. Objective is to make the transition from system-level design in Figure A.14 towards the DSL, and the validation versus the requirements seamless. To achieve this objective, the overall SICYPHOS framework, in addition to the SysML code transformations that are the focus of this work, provide the following modeling libraries and tools:

- **Virtual Prototyping Library (VPLib)**, which provides building blocks for communication (ComLib) and automotive (AutoLib) domains in the modeling language SystemC.
- **Wireless TLM** provides a wireless propagation and communication model based on SystemC TLM extensions.
- **UncertaintyCoverage** is a tool for symbolic propagation of uncertainties (e.g. aging, tolerances) in CPS.
- **PowerCoverage** is a SystemC based framework to estimate power consumption and to create meaningful highlevel profiles for cross-layer energy optimization of distributed embedded systems.

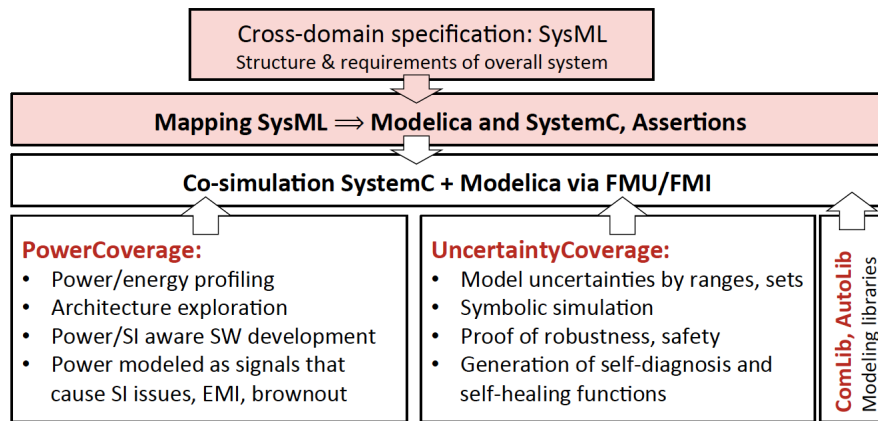


Figure A.15: SICYPHOS Framework [Waw+15]

Design Methodology. We assume that the overall system development starts with a cross-domain system specification using SysML. The code generators presented here then

generate the modules and interfaces for all subsystems in different domains. The output is a template of a virtual prototype of the HW/SW system (C++, SystemC), including templates for the network (SystemC Wireless TLM) and propagation modeling, and templates for modeling the physical environment in SystemC AMS or Modelica.

The generation of code templates for the code to be developed in e.g. C++, but as well of models of the environment allows developers to validate designs and– code immediately against the needs of the overall system. In addition, the PowerCoverage and UncertaintyCoverage tools allow estimation of power and accuracy/robustness.

The vision is to provide a significant, yet very flexible abstraction of Cyber-Physical Systems. This abstraction shall provide the necessary means to quickly and completely set up a simulation with alternating test cases. By having a formal specification of the system, it can be set up and designed with consistent semantics.

Furthermore, developers shall be enabled to integrate preexisting models or components without detailed knowledge. Of course functional methods and the IP have to be written by experts, but with pre-existing implementations and the appropriate ontologies, a ‘non-expert’ shall become capable of doing initial explorations of the architecture to get some early valuable feedback (power consumption, performance) of system behavior. If a simulation is executable in that way, the purpose is to use this information for wiser and effective design decisions in the early design phases.

Implementation. To achieve the objective, we intend to leverage a variety of methodologies in model-based and knowledge-based engineering. This paper focuses on the integration with SysML. To bridge the gap between the formal and graphical design entry and the domain specific languages we used the Acceleo code generator. SICYPHOS modeling with SysML contributes in six key aspects. This work presents the demonstration of the first two, while the other outlined aspects are work in progress.

1) IP block generation (ComLib)

For the generation of our modules, we followed a similar approach to the code generation in [Caf+13]. We generate basic functionalities from SysML models with the Acceleo code generator. However, we did not include a model-to-model transformation and we will examine if it is beneficial for our application. Acceleo was chosen because of its pragmatism, auto completion, and because the generation was more flexible compared to code generation within Enterprise Architect. Furthermore the integration with other modeling tools and Eclipse C++ yields a complete development experience. The ComLib library is implemented in SystemC-AMS and we thus generate code for this language. The elements translations are represented in Table 1. Additionally, we added our IP to the code generation. So far this is done sporadically from the template itself that also contains SystemC-AMS functions and code. Round-trip facilities are planned so that attributes and variable declarations can be brought back into the SysML model. Synchronization will then prevent that either model or code run out-of-date.

2) Wireless TLM

Wireless TLM is a framework to model wireless communication and radio propagation. For that purpose, it offers a set of SystemC modules and TLM interfaces:

- **node_base:** is the top-level API. Every top-level component with wireless connec-

tivity must be an extension of this class.

- **wtlm_module**: is the base class for the physical layer implementation. It provides the wireless interface itself. All *node_base* elements must have a *wtlm_module* that has to be extended by the user to include application-specific aspects, such as transceiver characteristics and the modulation and bitrate defined by the protocol used.
- **network_protocol**: is the base class for any network protocol between the physical wireless interface (*wtlm_module*) and the top-level application (*node_base*).

Therefore a SysML package has been defined that includes these three basic components, as shown in Figure A.16.

Work in Progress. 1) Cross-domain modeling with Modelica

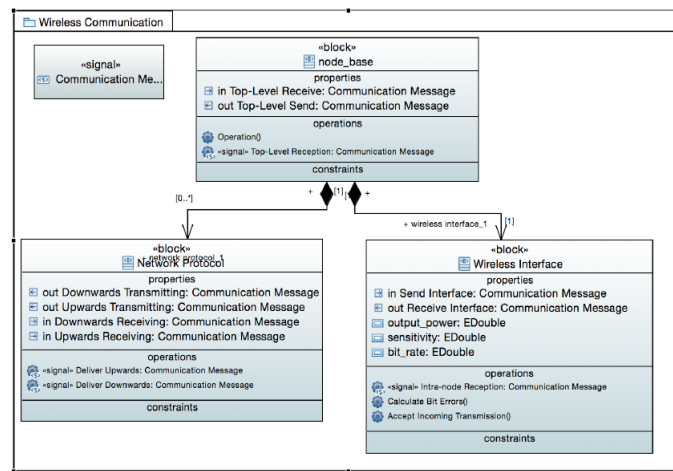


Figure A.16: Wireless Communication SysML Package [Waw+15]

In the simulation of Cyber-Physical Systems we deal with a variety of systems, domains, models of computation and environments. One language is often not sufficient to cover the heterogeneity of a system. Here we decided to integrate Modelica with SystemC / SystemC AMS respectively to provide a simulation of both, the nodes, sensors as well as mechanical structures, chemical processes and environmental influences. The coupling interface development is currently being finished and will be presented in other works. It is to be generated out of connected SysML Blocks. Necessary design decisions are abstracted.

2) Converter adaptation

When we consider the semantic adaptations in [Caf+13], they really represent just the most basic. Coupling modules just within SystemC/AMS/TLM can be complex and require additional context as to what and how it is coupled. The mathematics and functionality of MoC's has to be considered. Especially, work that is done in [Dam15] may be integrated. Amongst a variety of converters it presents a coupling of TLM with TDF by conversion.

3) Assertions

As already introduced, we use a language to specify assertions about system behavior. For instance, considering the use case with the control loop above. A simple assertion would be to check if the rising time to the reference value suffices, while, simultaneously, the settling error also stays within a certain boundary at a certain time. With Affine Arithmetic, ranges are calculated and propagated to estimate the fulfillment of the assertions. While this work is complete in itself, experience shows that developers are hesitant to work with such rather complex expressions. We intend to abstract these assertions to provide easy to handle semantics that can be integrated in a SysMLbased drag and drop design experience.

4) Power State Machines

SICYPHOS provides the infrastructure to estimate and track power consumption and create energy profiles that provide valuable and meaningful information across different domains. System-level power consumption estimator is implemented using finite state machines. Those state machines are the input for the power and energy profiler. State machines can therefore be specified using SysML and be used by the energy profilers to provide complex and abstract energy consumption estimations of software, communication, and even distributed tasks. Furthermore, state machines can also be used to assess the consistency of power consumption and to identify possible risks, such as brownout or crosstalk.

Use Cases. A. PWM Control Loop

As a demonstration for the modules in our ‘comlib’ library, we chose to model a simple control loop with SysML in Papyrus. The loop consists of a pwm, a PI controller, a difference module and a load. It regulates to a reference current source. The Block definition diagram of this example is shown in Figure A.17.

The block ‘T_PWMControlLoop’ is in this case the system and testbench, which con-

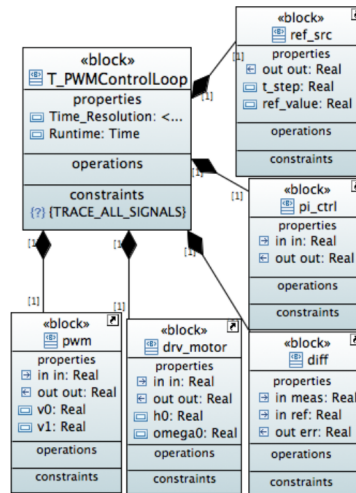


Figure A.17: Block Definition Diagram of Control Loop [Waw+15]

tains the control loop as subblocks. Parameters to be defined at this level are the ‘Time Resolution’, which determines the step width and thus accuracy of the simulation. ‘Run-time’ defines the total runtime of the simulation. The framework will also offer a variety of modeling facilities. The tracing of signals is a simple example that is used by SystemC-AMS itself and which we implemented as a constraint on the testbench with ‘TRACE_ALL_SIGNALS’. It ensures tracing of all signals in the testbench as well as storing trace data to an appropriate log file. Ports are defined as type real with their corresponding direction.

Through the reference source block, the simulation timestep ‘t_step’ and the actual reference current value ‘ref_value’ are adjusted. For the PWM just the high and low plateau values of the modulation are shown. It also offers the changing of periodic time / frequency and ramp time slope values. The load ‘drv_motor’ represents the degree of an opening of a throttle valve and is modeled via a resistance and inductance.

The implementation of the use case is shown in Figure A.18 within an internal block diagram. Parts are instances of their definition from the ‘blocks’ of the block definition diagram. Their ports are connected via SysML Connectors. In this example single signals can also be traced with the ‘TRACE_SIGNAL’ constraint. The constraint is allocated to a signal with a constraint link.

To bridge the gap between the formal definition in SysML and an executable simulation

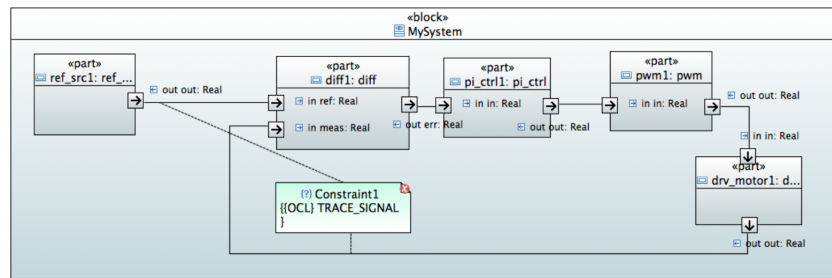


Figure A.18: Internal Block Diagram of Control Loop [Waw+15]

we used the Acceleo code generator as mentioned above. It is easily integrated into the design flow as a java project, but can also be a stand-alone application. Acceleo provides a mapping language that enables accessing the SysML and UML meta-models with their respective properties as well as OCL elements and constraints. A simple programming syntax yields the most important functionality. In case it is required, queries or Java Services can be written additionally. The mapping has been implemented for this test case.

Figure A.19 shows the mapping of the parts in the test case of Figure A.18 without the connections. Basically it is iterated through the packages, classes and properties in the UML model file and corresponding variables are accessed. OCL is used to filter the properties. Afterwards we run the simulation and obtain a characteristic PI control behavior. Figure A.20 shows this with traces of the reference value, the control error, as well as the measured current.


```

//Instantiations (unformatted)
[for (p:Package | aModel.eContents(Package))]
[if (p.name = 'Testbenches')]
[for (c:Class | p.eContents(Class))]
[for (parts:Property | c.eContents(Property))]
[parts.type.name/] [parts.name/]("["parts.name/]"
[for (p:Package | aModel.eContents(Package))]
[if (p.name = 'scp_comlib')]
[for (cl:Class | p.eContents(Class))]
[if parts.type.name = cl.name]
[for (property :Property | cl.ownedElement->select(p |
p.ocIsKindOf(Property)
and not p.ocIsKindOf(Port))), [property.name/]
[/for]
[/if]
[/for]
[/if]
[/for]):

```

Figure A.19: Acceleo Mapping for Module Instantiations [Waw+15]

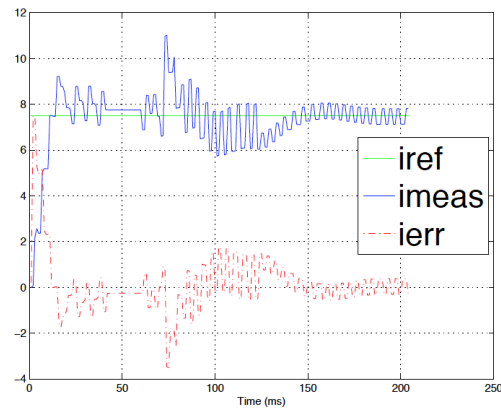


Figure A.20: Simulation Output [Waw+15]

B. Wireless Scenario

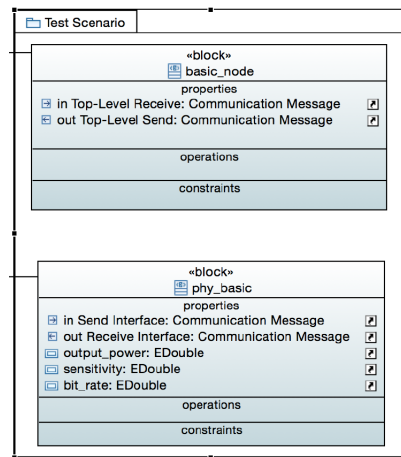


Figure A.21: SysML Package of the Test Scenario [Waw+15]

To setup a wireless scenario, the user must extend the Wireless TLM base classes with application specific methods and parameters. Figure A.21 shows a SysML package that contains specializations of the SysML blocks provided in the Wireless Communication package presented in Section III.B.2).

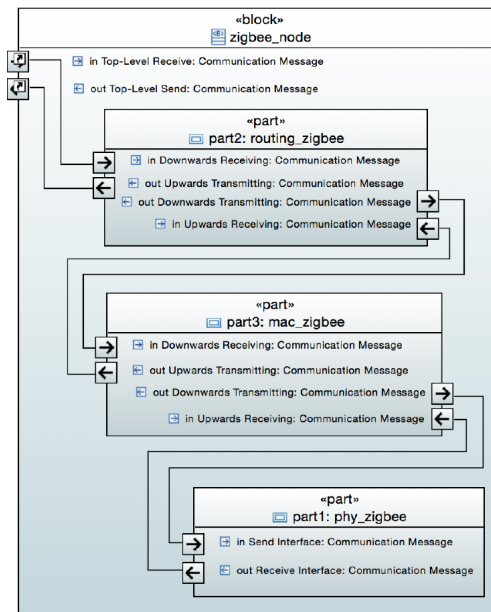


Figure A.22: Internal Block Diagram of a ZigBee node [Waw+15]

This way, the user can define the specific physical layer with the specific parameters and methods of the transceiver and protocol being used. The user can also define the network protocols.

Finally with the node_base specialization the user can implement the application and define the communication stack architecture. However, in order to define the communication stack, a SysML Internal Block Diagram (IBD) must be used, where all protocols are defined as parts of the top-level node block and the flow-ports are connected in the appropriate order. Figure 9 shows the IBD of a ZigBee node implementation. This way, the internal blocks of the node can be instantiated and the TLM sockets can be bound based on the SysML flow-port connectors."

List of Figures

1.1	Engineering 4.0 Overview ([Mat16], modified)	6
1.2	Ontology Example ([Hor11], modified)	7
1.3	SysML Motivation [San09]	8
1.4	SysML Overview [San09]	9
1.5	IoT Direction [Søn16]	10
1.6	Semantic Web Stack Overview [Wik18]	11
1.7	Web of Thought Vision [TL16]	12
2.1	Ontology Integration with Simulation Models [Gro+12]	17
2.2	The Extended Requirements Ontology [Li+15]	18
2.3	CONON Context Ontology [Wan+04]	19
2.4	3LConOnt Context Ontology [CFM19]	20
2.5	Stimulus-Sensor-Observation Pattern [Com+12]	21
2.6	VICINITY Core Ontology	22
2.7	Comparison between UML and OWL [KA08b]	25
2.8	MAPE-K Architecture [Gur+13]	31
2.9	Autonomous Agents Development [Wik19]	32
2.10	Autonomous Agents Vision ([Ver+09], modified)	32
3.1	Continuum of Expressiveness and Formality [UG04]	34
3.2	Reality and Ontology [GOS09]	37
3.3	Types and Categories of Ontologies	38
3.4	DKAP Methodology Structure [SF07]	44
3.5	Patterns for Identifying the Scope of Transitivity of Part-Whole Relations [Gui14]	46
3.6	Examples of Valid but Unintended Instances of the Organ Transplant Model [Gui14]	46
3.7	Example (A) and Counterexample (B) of Warranted Inference of Part-Whole Relation [Gui14]	47
3.8	A Four-Level Instantiation Chain in a Biological Domain. [Fon+18]	47
3.9	MLT* Basic Scheme Extended by a Domain Example [Fon+18]	48
3.10	The Structure and Occurrences of AP1 in Wikidata (Excerpt) [Joã+18]	49
3.11	Reasoning Constructs Part 1	49
3.12	Reasoning Constructs Part 2	50
3.13	Basic Formal Ontology Continuant Part [ASS15]	52

3.14	Part-Whole Relationships Taxonomy [Kee17]	53
3.15	Family of Languages/Logic supported by DOL with its Translations (Arrows) [Mos+14]	54
4.1	Scope of Tool AgilA [WG18]	57
4.2	Graphical User Interface of Students [WG18]	58
4.4	Initial Architecture of AgilA	58
4.3	Graphical User Interface of Author	59
4.5	Structural Hierarchy of the Engine via is-A and hasParts Relationships [WG18]	60
4.6	Overview of the Components of the Ontology [WG18]	60
4.7	Agenda for current Engine Instance [WG18]	63
4.8	Some Agenda Selection Criteria, also concerning the Engine [WG18]	63
4.9	Example of Conflict Resolution Rule [WG18]	64
4.10	Example of Control Rule [WG18]	64
4.11	Refined Distributed Architecture [WG18]	65
4.12	SWRL Constraints	66
5.1	Different Parties using GENIAL! Tool	68
5.2	GENIAL! Modular Ontology Suite	71
5.3	Formalization of Component Definition	74
5.4	Instantiation of a Property and its Unit	75
5.5	Instantiation of Ontology for Innovation for EFuse	76
5.6	Effect Chain Ontology in GENIAL	77
5.7	Class Hierarchy of the Solution Space of Hardware and Software Solutions	78
5.8	Restrictions on Solution Space of FunctionB	78
5.9	EFuse Prototype Showing Classes, Instances and Restrictions	79
5.10	SWRL Rule Chain to induce new Casings	80
5.11	HW / SW Allocation Ontology Prototype	81
5.12	Electric Fuse Ontology Model [Sun20]	82
5.13	Summary of InfoSec (left) and Artificial Intelligence (right) Ontology [Sun20]	84
5.14	Individuals of the Roadmap Ontology	84
5.15	Demographic Development Ontology Model	85
5.16	Instance Hierarchy of Hardware and Software Elements	88
5.17	Taxonomy of Hardware Elements	89
5.18	Interaction of Constraints and Knowledge Base	94
5.19	SysMD Dependencies Example in Tutorial	95
6.1	Research Directions Outlook	98
A.1	BFO 2.0 All Classes [Abh16]	100
A.2	GBO Core Part 1 Overview - Parts	102
A.3	More Detailed View on GBO Parts and Components	103

A.4	GBO Core Part 2 Overview - Properties and Functions	104
A.5	TBox Reasoning of Digital Filter System	108
A.6	Basic Sensor Types Taxonomy	109
A.7	An Instance of a Pulpissimo Controller as GBO Component	117
A.8	Some Software of the Pulpissimo Controller	117
A.9	Bended Graph with Inverse Relations	118
A.10	Graph with More Nodes	118
A.11	More Parts and Hardware Subparts	119
A.12	Ontology Service Loading Ontologies and Initializing Reasoner	120
A.13	Ontology Service Writing ABox Triples to Database (Excerpt)	121
A.14	SysML and domain-specific languages in the V diagram [Waw+15]	127
A.15	SICYPHOS Framework [Waw+15]	128
A.16	Wireless Communication SysML Package [Waw+15]	130
A.17	Block Definition Diagram of Control Loop [Waw+15]	131
A.18	Internal Block Diagram of Control Loop [Waw+15]	132
A.19	Acceleo Mapping for Module Instantiations [Waw+15]	133
A.20	Simulation Output [Waw+15]	133
A.21	SysML Package of the Test Scenario [Waw+15]	134
A.22	Internal Block Diagram of a ZigBee node [Waw+15]	134

List of Tables

4.1	Constraint Net of Use Case [WG18]	61
4.2	Fuel Consumption of Cars [WG18]	61
5.1	Comparison between the System Modeling Language and the Ontology Web Language	69
5.2	Basic Vocabulary - Part 1	73
5.3	Basic Vocabulary - Part 2	74
5.4	SWRL Rules to Deduce New Casing from Faster Algorithm	80
5.5	SWRL Rules for Roadmapping	87
A.1	Prefixes and Namespaces used within my Neural Net Accelerator Knowledge Base	99
A.2	Classes and its Definitions in GBO V0.2.3 - Part 1	105
A.3	Classes and its Definitions in GBO V0.2.3 - Part 2	106
A.4	Classes and its Definitions in GBO V0.2.3 - Part 3	107
A.5	A Selection of Classes and its Definitions in HW / SW Domain Ontology V0.1.1	110

Abbreviations

<i>ADC</i>	Analog to Digital Converter
<i>AgilA</i>	Agile Automotive
<i>AMS</i>	Analog Mixed Signal
<i>BFO</i>	Basic Formal Ontology
<i>CPS</i>	Cyber Physical System
<i>DAC</i>	Digital to Analog Converter
<i>DSL</i>	Domain Specific Language
<i>FOL</i>	First Order Logic
<i>GBO</i>	GENIAL! Basic Ontology
<i>GENIAL</i>	Common Electronics Roadmap for Innovations in the Automotive Value Chain
<i>GUI</i>	Graphical User Interface
<i>HW/SW</i>	Hardware/Software
<i>IoT</i>	Internet of Things
<i>IP</i>	Intellectual Property
<i>ISO</i>	International Organisation for Standardization
<i>JSON</i>	JavaScript Object Notation
<i>MAPE – K</i>	Monitor Analyze Plan Execute Knowledge
<i>MLT</i>	Multi-Level Theory
<i>MoC</i>	Model of Computation
<i>MOF</i>	Meta Object Facility
<i>NoSQL</i>	Not only Structured Query Language
<i>OCL</i>	Object Constraint Language
<i>OEM</i>	Original Equipment Manufacturer
<i>OWL</i>	Ontology Web Language
<i>OWLAPI</i>	Ontology Web Language Application Programming Interface
<i>PI</i>	Proportional Integral (Controller)
<i>PID</i>	Proportional Integral Differentiate
<i>PLL</i>	Phase Locked Loop
<i>PWM</i>	Pulse Width Modulation
<i>RDFS</i>	Resource Description Framework Schema
<i>SICYPHOS</i>	SImulatiOn and SYnthesis of CYber PHysical Systems and ApplicatiOnS

<i>SoC</i>	System on Chip
<i>SPARQL</i>	SPARQL Protocol And RDF Query Language
<i>SQWRL</i>	Semantic Query-Enhanced Web Rule Language
<i>SWRL</i>	Semantic Web Rule Language
<i>SysML</i>	System Modeling Language
<i>TDF</i>	Timed Data Flow
<i>TLM</i>	Transaction Level Modeling
<i>TLO</i>	Top Level Ontology
<i>UML</i>	Unified Modeling Language
<i>VHDL</i>	Very High speed integrated Hardware Description Language
<i>XMI</i>	eXtensible Markup Interchange
<i>XSLT</i>	eXtensible Stylesheet Language Transformation

Literature

- [Abh16] Kumar Abhishek. “An Ontology based Decision support for Tuberculosis Management and Control in India”. In: *International Journal of Engineering and Technology* 8 (Dec. 2016), pp. 2860–2877. DOI: [10.21817/ijet/2016/v8i6/160806247](https://doi.org/10.21817/ijet/2016/v8i6/160806247).
- [And07] Dietmar Zaefferer Andrea C. Schalley, ed. *Ontolinguistics: How Ontological Status Shapes the Linguistic Coding of Concepts*. Trends in Linguistics. Studies and Monographs [TiLSM], 176. De Gruyter Mouton, 2007.
- [ASS15] Robert Arp, Barry Smith, and Andrew D. Spear. *Building Ontologies with Basic Formal Ontology*. The MIT Press, 2015. ISBN: 0262527812, 9780262527811.
- [BAS17] Lamyaa EL BASSITI. “Generic Modular Ontology for Innovation Domain. A Key Pillar Towards “Innovation Interoperability””. In: *Journal of Entrepreneurship, Management and Innovation* 13.2 (2017), pp. 105–126. DOI: [10.7341/20171325](https://doi.org/10.7341/20171325). URL: <https://ideas.repec.org/a/aae/journal/v13y2017i2p105-126.html>.
- [BBM14] Youcef Belgueliel, Mustapha Bourahla, and Brik Mourad. “Towards an Ontology for UML State Machines”. In: *Lecture Notes on Software Engineering* (Jan. 2014), pp. 116–120. DOI: [10.7763/LNSE.2014.V2.106](https://doi.org/10.7763/LNSE.2014.V2.106).
- [Be18] Kenneth Baclawski and Mike Bennet et.al. *Ontology Summit 2018 Communiqué*. 2018. URL: <https://ken.baclawski.com/pub/2018/30/public.pdf>.
- [Ber+20] P. P. Bernardo et al. “UltraTrail: A Configurable Ultralow-Power TC-ResNet AI Accelerator for Efficient Keyword Spotting”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 4240–4251. DOI: [10.1109/TCAD.2020.3012320](https://doi.org/10.1109/TCAD.2020.3012320).
- [BHR95] John A. Bateman, Renate Henschel, and Fabio Rinaldi. *Generalized Upper Model 2.0: documentation*. Tech. rep. Darmstadt, Germany: GMD/Institut für Integrierte Publikations- und Informationssysteme, 1995. URL: <http://www.darmstadt.gmd.de/publish/komet/gen-um/newUM.html>.
- [Bir96] Hans-Jürgen Sebastian Birger Funke. “Knowledge-based model building with KONWERK”. In: *International Institute for Applied Systems Analysis*. 1996.

- [Blo+11] T. Blochwitz et al. “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *In Proceedings of the 8th International Modelica Conference*. 2011.
- [BLT10] C. Brooks, E. A. Lee, and S. Tripakis. “Exploring models of computation with Ptolemy II”. In: *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2010, pp. 331–332. DOI: [10.1145/1878961.1879020](https://doi.org/10.1145/1878961.1879020).
- [Boh02] David Bohm. *Wholeness and the Implicate Order*. Routledge; 1 edition (November 17, 2002), 2002.
- [Bra11] Keyla Guimarães Macharet Brasil. “Automatic Translation of SysML Models to SystemC Executable Specification”. In: 2011.
- [Caf+13] D. C. Café et al. “Multi-paradigm semantics for simulating SysML models using SystemC-AMS”. In: *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*. 2013, pp. 1–8. ISBN: 1636-9874.
- [Car+12] T. Cardoso et al. “Communication software synthesis from UML-ESL models”. In: *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*. 2012, pp. 1–6. DOI: [10.1109/SBCCI.2012.6344449](https://doi.org/10.1109/SBCCI.2012.6344449).
- [Car88] Luca Cardelli. “Structural Subtyping and the Notion of Power Type”. In: *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*. ACM Press, Jan. 1988, pp. 70–79. URL: <https://www.microsoft.com/en-us/research/publication/structural-subtyping-and-the-notion-of-power-type/>.
- [Cas+10] Verónica Castañeda et al. “The Use of Ontologies in Requirements Engineering”. In: *Glob J Res Eng* 10 (Jan. 2010).
- [CCB06] Jon Curtis, John Cabral, and David Baxter. “On the Application of the Cyc Ontology to Word Sense Disambiguation.” In: Jan. 2006, pp. 652–657.
- [CFG03] Oscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. “Methodologies, tools and languages for building ontologies. Where is their meeting point?” In: *Data Knowl. Eng.* 46.1 (July 2003), pp. 41–64. ISSN: 0169-023X. DOI: [10.1016/S0169-023X\(02\)00195-7](https://doi.org/10.1016/S0169-023X(02)00195-7).
- [CFM19] Oscar Cabrera, Xavier Franch, and Jordi Marco. “3LConOnt: A Three-Level Ontology for Context Modelling in Context-Aware Computing”. In: *Software & Systems Modeling* 18.2 (Apr. 2019), pp. 1345–1378. ISSN: 1619-1374. DOI: [10.1007/s10270-017-0611-z](https://doi.org/10.1007/s10270-017-0611-z). URL: <https://doi.org/10.1007/s10270-017-0611-z>.
- [Che+04] H. Chen et al. “SOUPA: standard ontology for ubiquitous and pervasive applications”. In: Sept. 2004, pp. 258–267. ISBN: 0-7695-2208-4. DOI: [10.1109/MOBIQ.2004.1331732](https://doi.org/10.1109/MOBIQ.2004.1331732).

- [Com+12] Michael Compton et al. “The SSN ontology of the W3C semantic sensor network incubator group”. In: *Journal of Web Semantics* 17 (2012), pp. 25–32. ISSN: 1570-8268. DOI: <https://doi.org/10.1016/j.websem.2012.05.003>. URL: <http://www.sciencedirect.com/science/article/pii/S1570826812000571>.
- [Cue+18] Javier Cuenca et al. “Towards Cognitive Cities in the Energy Domain”. In: *Designing Cognitive Cities* (2018).
- [Dam15] Markus Damm. “Crossing Modeling Paradigms in System Models”. Ph.D. Thesis. TU Wien, March 2015.
- [Dan15] Clemens Danninger. *Using Constraint Solvers to Find Valid Software Configurations*. 2015. (Visited on 2018).
- [DCV21] Beniamino Di Martino, Luigi Colucci Cante, and Salvatore Venticinqué. “An Ontology Framework for Evaluating E-mobility Innovation”. In: *Complex, Intelligent and Software Intensive Systems*. Ed. by Leonard Barolli, Aneta Poniszewska-Maranda, and Tomoya Enokido. Cham: Springer International Publishing, 2021, pp. 520–529. ISBN: 978-3-030-50454-0.
- [DJS15] Nikil Dutt, Axel Jantsch, and Santanu Sarma. “Self-Aware Cyber-Physical Systems-on-Chip”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ICCAD ’15. Austin, TX, USA: IEEE Press, 2015, pp. 46–50. ISBN: 978-1-4673-8389-9. URL: <http://dl.acm.org/citation.cfm?id=2840819.2840826>.
- [DLV12] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli. “Modeling Cyber Physical Systems”. In: *Proceedings of the IEEE* 100.1 (2012), pp. 13–28. DOI: [10.1109/JPROC.2011.2160929](https://doi.org/10.1109/JPROC.2011.2160929).
- [Dup17] Philippe Dupuy. *Improving the automotive power distribution architecture - Power Electronics*. [Online; accessed 31. Jan. 2020]. Dec. 2017. URL: <https://www.powerelectronicsnews.com/technology/improving-the-automotive-power-distribution-architecture>.
- [Ehm+19] Hans Ehm et al. “Digital Reference – A Semantic Web for Semiconductor Manufacturing and Supply Chains Containing Semiconductors”. In: Dec. 2019, pp. 2409–2418. DOI: [10.1109/WSC40007.2019.9004831](https://doi.org/10.1109/WSC40007.2019.9004831).
- [Eri+18] Henrik Eriksson et al. “Simulation Modeling using Protégé”. In: (June 2018).
- [Fah+18] Hossam Faheem et al. “Ontology Design for Solving Computationally- Intensive Problems on Heterogeneous Architectures”. In: *Sustainability* 10 (Feb. 2018), p. 441. DOI: [10.3390/su10020441](https://doi.org/10.3390/su10020441).
- [FKV14] Stefan Feldmann, Konstantin Kernschmidt, and Birgit Vogel-Heuser. “Combining a SysML-based Modeling Approach and Semantic Technologies for Analyzing Change Influences in Manufacturing Plant Models”. In: *Procedia CIRP* 17 (2014). Variety Management in Manufacturing, pp. 451–456. ISSN: 2212-8271. DOI: <https://doi.org/10.1016/j.procir.2014.01.140>. URL: <http://www.sciencedirect.com/science/article/pii/S2212827114004181>.

- [FM11] Michael Feld and Christian Müller. “The Automotive Ontology: Managing Knowledge Inside the Vehicle and Sharing it Between Cars”. In: *Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications. International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI-11)*, November 29-December 2, Salzburg, Austria. ACM, Nov. 2011.
- [Fon+18] Claudenir Fonseca et al. “Multi-Level Conceptual Modeling: From a Formal Theory to a Well-Founded Language”. In: Oct. 2018.
- [GCS90] Andreas Günter, Roman Cunis, and Ingo Syska. “Separating Control from Structural Knowledge in Construction Expert Systems”. In: *Proceedings of the 3rd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - Volume 2*. IEA/AIE ’90. Charleston, South Carolina, USA: ACM, 1990, pp. 601–610. ISBN: 0-89791-372-8. DOI: [10.1145/98894.98904](https://doi.org/10.1145/98894.98904). URL: <http://doi.acm.org/10.1145/98894.98904>.
- [GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. “What Is an Ontology?”. In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. International Handbooks on Information Systems. Springer, 2009, pp. 1–17. ISBN: 978-3-540-92673-3. URL: <http://dblp.uni-trier.de/db/series/ihi/hoo2009.html#GuarinoOS09>.
- [Gra09] Henson Graves. “Integrating SysML and OWL”. In: *Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529*. OWLED’09. Chantilly, VA: CEUR-WS.org, 2009, pp. 117–124. URL: <http://dl.acm.org/citation.cfm?id=2890046.2890059>.
- [Gri+21] Christoph Grimm et al. “APPEL - AGILA ProPerty and Dependency Description Language”. In: *MBMV 2021 - 24. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*. Mar. 2021.
- [Gro+12] Katarina Grolinger et al. “Ontology-based Representation of Simulation Models”. In: *SEKE*. 2012.
- [Gru95] T. R. Gruber. “Toward principles for the design of ontologies used for knowledge sharing”. In: vol. Vol. 43. *International Journal of Human-Computer Studies*, Nov. 1995, pp. 907-928.
- [GSA15] Amelie Gyrard, Martin Serrano, and Ghislain A. Atemezing. “Semantic web methodologies, best practices and ontology engineering applied to Internet of Things”. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)* (Dec. 2015), pp. 412–417. DOI: [10.1109/WF-IoT.2015.7389090](https://doi.org/10.1109/WF-IoT.2015.7389090).
- [Gui05] Giancarlo Guizzardi. “Ontological foundations for structural conceptual models”. English. PhD thesis. University of Twente, Oct. 2005. ISBN: 90-75176-81-3.

- [Gui14] Giancarlo Guizzardi. “Ontological Patterns, Anti-Patterns and Pattern Languages for Next-Generation Conceptual Modeling”. In: Oct. 2014, pp. 13–27. DOI: [10.1007/978-3-319-12206-9_2](https://doi.org/10.1007/978-3-319-12206-9_2).
- [Gur+13] Levent Gurgun et al. “Self-aware Cyber-physical Systems and Applications in Smart Buildings and Cities”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '13. Grenoble, France: EDA Consortium, 2013, pp. 1149–1154. ISBN: 978-1-4503-2153-2. URL: <http://dl.acm.org/citation.cfm?id=2485288.2485566>.
- [GW18] Christoph Grimm and Frank Wawrzik. “Ein wissensbasierter Ansatz für Entwicklung 4.0”. In: *21. Workshop „Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen”*. 2018.
- [GWZ19] Christoph Grimm, Frank Wawrzik, and Carna Zivkovic. “An Approach for agile, distributed Development of Cyber-Physical Systems of Systems”. In: *22. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. VDE-Verlag, Mar. 2019. ISBN: 978-3-8007-4946-1.
- [Hal+18] Armin Haller et al. “The SOSA/SSN ontology: a joint Web and OGC standard specifying the semantics of sensors observations actuation and sampling”. In: *Semantic Web Journal*. Vol. Vol. 1. IOS Press. 2018, pp. 1–19.
- [Hen08] N. Henze. *Semantic Web The Proof and Trust Layers of the Semantic Web*. http://www.kbs.uni-hannover.de/Lehre/semweb07/10_proof_trust.pdf. Jan. 2008.
- [Her+12] F. Herrera et al. “Enhanced IP-XACT Platform Descriptions for Automatic Generation from UML/MARTE of Fast Performance Models for DSE”. In: *2012 15th Euromicro Conference on Digital System Design*. 2012, pp. 692–699. DOI: [10.1109/DSD.2012.51](https://doi.org/10.1109/DSD.2012.51).
- [HH06] H. Herre and B. Heller. “Semantic Foundations of Medical Information Systems Based on Top-level Ontologies”. In: *Know.-Based Syst.* 19.2 (June 2006), pp. 107–115. ISSN: 0950-7051. DOI: [10.1016/j.knosys.2005.10.002](https://doi.org/10.1016/j.knosys.2005.10.002). URL: <http://dx.doi.org/10.1016/j.knosys.2005.10.002>.
- [Hor11] Matthew Horridge. *A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.3*. http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf. Mar. 2011.
- [HRC87] R. Harjani, R. A. Rutenbar, and L. R. Carley. “A Prototype Framework for Knowledge-Based Analog Circuit Synthesis”. In: *24th ACM/IEEE Design Automation Conference*. 1987, pp. 42–49. ISBN: 0738-100X. DOI: [10.1145/37888.37894](https://doi.org/10.1145/37888.37894).
- [Isa15] Joanna Isabelle Olszewska. *UML Activity Diagrams for OWL Ontology Building*. Jan. 2015.

- [JB17] Martin Jaiser and Manfred Brandl. *Smart fuse reduces cost, weight of automotive wiring harness*. [Online; accessed 31. Jan. 2020]. Sept. 2017. URL: <https://www.eenewsautomotive.com/design-center/smart-fuse-reduces-cost-weight-automotive-wiring-harness>.
- [JDR17] A. Jantsch, N. Dutt, and A. M. Rahmani. “Self-Awareness in Systems on Chip A Survey”. In: *IEEE Design & Test* 34.6 (2017), pp. 8–26. DOI: [10.1109/MDAT.2017.2757143](https://doi.org/10.1109/MDAT.2017.2757143).
- [Jen12] *Semantically-rigorous systems engineering using SysML and OWL*. Vol. 5th International Workshop on System & Concurrent Engineering for Space Applications SECESA 2012. Lisbon, Portugal: Pasadena, CA : Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2012., Oct. 2012.
- [JKR11] Vaibhav Jain, Anshul Kumar, and Preeti Ranjan Panda. *A SysML Profile for Development and Early Validation of TLM 2.0 Models*. June 2011.
- [Joã+18] Paulo João et al. “Multi-Level Conceptual Modeling: Theory and Applications”. In: Oct. 2018.
- [Joã+19] Paulo João et al. “Preserving Multi-Level Semantics in Conventional Two-Level Modeling Techniques”. In: (Aug. 2019).
- [JWS01] Zhao Junchao, Chen Weiliang, and Wei Shaojun. “Parameterized IP core design”. In: *ASICON 2001. 4th International Conference on ASIC Proceedings (Cat. No.01TH8549)*. 2001, pp. 744–747. DOI: [10.1109/ICASIC.2001.982670](https://doi.org/10.1109/ICASIC.2001.982670).
- [KA08a] C. Maria Keet and Alessandro Artale. “Representing and Reasoning over a Taxonomy of Part-whole Relations”. In: *Appl. Ontol.* 3.1-2 (Jan. 2008), pp. 91–110. ISSN: 1570-5838. URL: <http://dl.acm.org/citation.cfm?id=2674733.2674738>.
- [KA08b] Kilian Kiko and Colin Atkinson. “A Detailed Comparison of UML and OWL”. In: 2008.
- [Kee11] C. Maria Keet. “The Use of Foundational Ontologies in Ontology Development: An Empirical Assessment”. In: *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part I*. ESWC’11. Heraklion, Crete, Greece: Springer-Verlag, 2011, pp. 321–335. ISBN: 978-3-642-21033-4. URL: <http://dl.acm.org/citation.cfm?id=2008892.2008920>.
- [Kee17] Maria C. Keet. “A note on the compatibility of part-whole relations with foundational ontologies”. In: *Proceedings of FOUST-II: 2nd Workshop on Foundational Ontology, Joint Ontology Workshops 2017*. Bolzano, Italy, Department of Computer Science, University of Cape Town, Sept. 2017.
- [Kee18] Maria C. Keet. *An Introduction to Ontology Engineering*. College Publications, 2018.

- [Klo+18] Benjamin Klotz et al. “VSSo: A Vehicle Signal and Attribute Ontology”. In: Oct. 2018.
- [KP11] Johannes Keizer and Valeria Pesce. *Semantic Interoperability*. [Online; accessed 31. Jan. 2020]. Aug. 2011. URL: <http://www.fao.org/docs/eims/upload/295352/KeizerPesce%20Think%20Piece%20Interoperability.pdf>.
- [Küh01] Christian Kühn. “Vergleich unterschiedlicher Konfigurationsmethoden im Hinblick auf die Nutzbarkeit von Wissens über das Zustandsverhalten der Konfigurationsobjekte”. In: *Proceedings of the KI-2001 Workshop AI in Planning, Scheduling, Configuration and Design, PuK*. Wien, 2001.
- [Kul+18] Boonserm Kulvatunyou et al. “The Industrial Ontologies Foundry Proof-of-Concept Project: IFIP WG 5.7 International Conference, APMS 2018, Seoul, Korea, August 26-30, 2018, Proceedings, Part II”. In: Aug. 2018, pp. 402–409. ISBN: 978-3-319-99706-3. DOI: [10.1007/978-3-319-99707-0_50](https://doi.org/10.1007/978-3-319-99707-0_50).
- [LFB96] Jinxin Lin, Mark S. Fox, and Taner Bilgic. “A Requirement Ontology for Engineering Design”. In: *Concurrent Engineering* 4.3 (1996), pp. 279–291. DOI: [10.1177/1063293X9600400307](https://doi.org/10.1177/1063293X9600400307). eprint: <https://doi.org/10.1177/1063293X9600400307>. URL: <https://doi.org/10.1177/1063293X9600400307>.
- [Li+15] Feng-Lin Li et al. “From Stakeholder Requirements to Formal Specifications Through Refinement”. In: Mar. 2015. DOI: [10.1007/978-3-319-16101-3_11](https://doi.org/10.1007/978-3-319-16101-3_11).
- [LS19] Jobst Landgrebe and Barry Smith. *Making AI meaningful again*. Jan. 2019.
- [Mat16] Peter Gabriel Matthias Künzel Jens Schulz. *Engineering 4.0 - Grundzüge eines Zukunftsmodells*. https://vdivde-it.de/sites/default/files/document/Engineering_40_Grundzuege-eines-Zukunftsmodells-2016.pdf. July 2016.
- [McC10] Dave McComb. “Gist: The minimalist upper ontology (abstract)”. In: *Semantic Technology Conference*. San Francisco, USA, 21-25 June 2010.
- [Mol+14] Javier Moreno Molina et al. *Model Based Design of Distributed Embedded Cyber Physical Systems*. Jan. 2014.
- [Mos+14] Till Mossakowski et al. “The Distributed Ontology, Modeling and Specification Language”. In: (Jan. 2014).
- [MYM07] Jan Morbach, Aidong Yang, and Wolfgang Marquardt. “OntoCAPE—A large-scale ontology for chemical process engineering”. In: *Engineering Applications of Artificial Intelligence* 20.2 (2007). Special Issue on Applications of Artificial Intelligence in Process Systems Engineering, pp. 147–161. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2006.06.010>. URL: <http://www.sciencedirect.com/science/article/pii/S0952197606001199>.
- [Nev+13] Olga Nevzorova et al. *Bringing Math to LOD: A Semantic Publishing Platform Prototype for Scientific Collections in Mathematics*. Oct. 2013.

- [NM19] Natalya Noy and Deborah McGuinness. *What is an ontology and why we need it*. [Online; accessed 31. Jan. 2020]. Sept. 2019. URL: https://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.
- [NP01] Ian Niles and Adam Pease. "Towards a Standard Upper Ontology". In: Jan. 2001, pp. 2–9. DOI: [10.1145/505168.505170](https://doi.org/10.1145/505168.505170).
- [OGW95] P. Oehler, G. Grimm, and K. Waldschmidt. "KANDIS-a tool for construction of mixed analog/digital systems". In: *Proceedings of EURO-DAC. European Design Automation Conference*. 1995, pp. 14–19. DOI: [10.1109/EURDAC.1995.527383](https://doi.org/10.1109/EURDAC.1995.527383).
- [Pen+06] J. Peng et al. "Automatic generation of transaction level models for rapid design space exploration". In: *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '06)*. 2006, pp. 64–69. DOI: [10.1145/1176254.1176272](https://doi.org/10.1145/1176254.1176272).
- [Pet96] Klaus Waldschmidt Peter Oehler. "A Knowledge Based System Level Construction Methodology". In: *The Second World Conference on Integrated Design Process Technology*. Austin Texas, USA, 1996.
- [Pov18] García-Castro R. Poveda-Villalón M. "Extending the SAREF ontology for building devices and topology". In: *6th Linked Data in Architecture and Construction Workshop (LDAC 2018)*. London, UK, 19-21 June 2018 2018.
- [PSM08] Jinsoo Park, Kimoon Sung, and Sewon Moon. *Developing Graduation Screen Ontology Based on the METHONTOLOGY Approach*. Vol. 2. IEEE, Sept. 2008. DOI: [10.1109/NCM.2008.215](https://doi.org/10.1109/NCM.2008.215).
- [Rec+06] Alan Rector et al. *Common Errors in OWL*. [Online; accessed 31. Jan. 2020]. June 2006. URL: https://protege.stanford.edu/conference/2004/slides/6.1%7B%5C_%7DHorridge%7B%5C_%7DCommonErrorsInOWL.pdf.
- [REM22] Nour Ramzy, Hans Ehm, and Patrick Moder. "Investigating Semantic Web as Enabler for Semiconductor Supply Chain Collaboration". In: Feb. 2022.
- [Rie+09] Christoph Riedl et al. "An Idea Ontology for Innovation Management". In: *International Journal on Semantic Web and Information Systems (IJSWIS)* 5.4 (2009), pp. 1–18. DOI: [10.4018/jswis.2009100101](https://doi.org/10.4018/jswis.2009100101). URL: <http://home.in.tum.de/~riedlc/res/RiedlEtAl2010.pdf>.
- [Run06] Wolfgang Runte. "YACS: Ein hybrides Framework für Constraint-Solver zur Unterstützung wissensbasierter Konfigurierung". MA thesis. 2006.
- [San09] Rick Steiner Sanford Friedenthal Alan Moore. *OMG SysML Tutorial*. <http://www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf>. Sept. 2009.
- [Sch18] Stefan Schulz. "The Role of Foundational Ontologies for Preventing Bad Ontology Design". In: Dec. 2018.

- [SF07] MD B. Sarder and Susan Ferreira. “Developing Systems Engineering Ontologies”. In: *2007 IEEE International Conference on System of Systems Engineering* (Apr. 2007), pp. 1–6. DOI: [10.1109/SYBOSE.2007.4304237](https://doi.org/10.1109/SYBOSE.2007.4304237).
- [SGW21] Siva D. Chandrasekaran Saratha, Christoph Grimm, and Frank Wawrzik. “A Digital Twin with Runtime-Verification for Industrial Development-Operation Integration”. In: *2021 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. 2021, pp. 1–9. DOI: [10.1109/ICE/ITMC52061.2021.9570222](https://doi.org/10.1109/ICE/ITMC52061.2021.9570222).
- [Sie+11] Katja Siegemund et al. “Towards Ontology-Driven Requirements Engineering”. In: Oct. 2011.
- [Søn16] David Sønstebo. *Honest Data - Ensuring Data Integrity*. <https://blog.iota.org/honest-data-f4e25bdac5ad>. Aug. 2016.
- [SSS05] S. Sarkar, S. Shinde, and C. G. Subash. “An Effective IP Reuse Methodology for Quality System-on-Chip Design”. In: *2005 International Symposium on System-on-Chip*. 2005, pp. 104–107. DOI: [10.1109/ISSOC.2005.1595655](https://doi.org/10.1109/ISSOC.2005.1595655).
- [Sta07] Steffen Staab. *What makes you think that? The Semantic Web’s Proof Layer*. <https://sites.ualberta.ca/~reformat/ece720w2012/papers/IEEEExplore-8.pdf>. 2007.
- [Sta14] OMG Standards. *Object Constraint Language (OCL) specification*. formal/2014-02-03. 2014.
- [Sta15] OMG Standards. *System Modeling Language (SysML) specification*. formal/15-06-03. 2015.
- [Sun20] Vishnu Vardhan Sundarrajan. “Towards a True Microelectronic and Automotive Digital Roadmap”. MA thesis. TU Kaiserslautern, Mar. 2020.
- [TL16] Mariapina Trunfio and Maria Lucia. “Toward Web 5.0 in Italian Regional Destination Marketing”. In: *Symphonya. Emerging Issues in Management* 2 (Jan. 2016), pp. 60–75. DOI: [10.4468/2016.2.07trunfio.dellalucia](https://doi.org/10.4468/2016.2.07trunfio.dellalucia).
- [UG04] Michael Uschold and Michael Grüninger. “Ontologies and Semantics for Seamless Connectivity.” In: *SIGMOD Record* 33 (Dec. 2004), pp. 58–64. DOI: [10.1145/1041410.1041420](https://doi.org/10.1145/1041410.1041420).
- [Ver+09] Ovidiu Vermesan et al. “Internet of Things Strategic Research Roadmap”. In: (Jan. 2009).
- [Wan+04] Xiao Hang Wang et al. “Ontology Based Context Modeling and Reasoning Using OWL”. In: *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. PERCOMW ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 18–. ISBN: 0-7695-2106-1. URL: <http://dl.acm.org/citation.cfm?id=977405.978618>.
- [Waw+] Frank Wawrzik et al. “Ontology-Based Roadmapping of Automotive Systems”. To be submitted to *IEEE Transactions on Software Engineering*.

- [Waw+15] F. Wawrzik et al. “Modeling and simulation of Cyber-Physical Systems with SICYPHOS”. In: *2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2015, pp. 1–6. DOI: [10.1109/DTIS.2015.7127375](https://doi.org/10.1109/DTIS.2015.7127375).
- [Waw17] Frank Wawrzik. *Whitepaper - Semantic Technologies - A Comparison*. 2017.
- [WG18] Frank Wawrzik and Christoph Grimm. “A Knowledge-based Approach for Configuring Automobile Design and Development”. In: *Forum on Specification and Design Languages (FDL)*. Munich, Germany, 2018.
- [Wik18] Wikipedia. *Semantic Web*. https://en.wikipedia.org/wiki/Semantic_Web. 2018.
- [Wik19] Wikipedia. *Intelligent Agent*. https://en.wikipedia.org/wiki/Intelligent_agent. 2019.
- [WL21] Frank Wawrzik and Andreas Lober. “A Reasoner-Challenging Ontology from the Microelectronics Domain”. In: *ISWC 2021 The 20th International Semantic Web Conference - Semantic Reasoning Evaluation Challenge (Sem-REC)*. Oct. 2021.
- [WMG14] F. Wawrzik, J. M. Molina, and C. Grimm. “A concept for design of embedded systems at semantic level”. In: *Proceedings of the 2014 Forum on Specification and Design Languages (FDL)*. Vol. 978-2-9530504-9-3. 2014, pp. 1–4. ISBN: 1636-9874. DOI: [10.1109/FDL.2014.7119352](https://doi.org/10.1109/FDL.2014.7119352).
- [Woo02] Michael Wooldridge. *Personal Agents*. <http://semantisches-web.net/web-30/personal-agents/>. 2002.
- [WR14] Bernhard Nebel Wolfram Burgard and Martin Riedmiller. *Foundations of Artificial Intelligence, 8. Machine Learning Lecture Material*. 2014.
- [WS08] Jules White and Douglas C. Schmidt. *Automated Configuration of Component-based Distributed Real-time and Embedded Systems from Feature Models*. 2008.
- [Xu+13] Nan Xu et al. “CACOnt: A Ontology-Based Model for Context Modeling and Reasoning”. In: 347-350 (Mar. 2013).
- [ZL12] Jesper Zedlitz and Norbert Luttenberger. “Transforming Between UML Conceptual Models and OWL 2 Ontologies”. In: Nov. 2012.
- [ZT13] Vladimir Zdraveski and Dimitar Trajanov. *VHDL IP CORES ONTOLOGY*. Apr. 2013.

Confirmation of dissertation fee

I have transferred the required dissertation fee to the Landeshochschulkasse Mainz (142€).

Kaiserslautern, April 6, 2022



Frank Wawrzik

References of publication list from the author

- [1] Frank Wawrzik, Christoph Grimm, Jack Martin, and Peter Neumann. Ontology-based roadmapping of automotive systems. To be submitted to IEEE Transactions on Software Engineering.
- [2] Frank Wawrzik and Andreas Lober. A reasoner-challenging ontology from the microelectronics domain. In *ISWC 2021 The 20th International Semantic Web Conference - Semantic Reasoning Evaluation Challenge (Sem-REC)*, 10 2021.
- [3] Frank Wawrzik and Christoph Grimm. A hardware / software allocation core ontology for collaboration along the value chain. In *IEEE 17th International Conference on Automation Science and Engineering*, 08 2021.
- [4] Siva D. Chandrasekaran Saratha, Christoph Grimm, and Frank Wawrzik. A digital twin with runtime-verification for industrial development-operation integration. In *2021 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–9, 2021.
- [5] Christoph Grimm, Frank Wawrzik, Alexander Louis-Ferdinand Jung, Konstantin Lübeck, Sebastian Post, Johannes Koch, and Oliver Bringmann. Appel - agila property and dependency description language. In *MBMV 2021 - 24. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 03 2021.
- [6] Frank Wawrzik and Christoph Grimm. Ontology design for microelectronics with roadmapping (work-in-progress). In *MBMV 2021 - 24. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, 03 2021.
- [7] Frank Wawrzik. Whitepaper - ontology research 2020 state of the art.
- [8] Christoph Grimm, Frank Wawrzik, and Carna Zivkovic. An approach for agile, distributed development of cyber-physical systems of systems. In *22. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. VDE-Verlag, March 2019.
- [9] Christoph Grimm and Frank Wawrzik. Ein wissensbasierter Ansatz für Entwicklung 4.0. In *21. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, March 2018.
- [10] Frank Wawrzik and Christoph Grimm. A knowledge-based approach for configuring automobile design and development. In *Forum on Specification and Design Languages*, Munich, Germany, 2018.
- [11] Frank Wawrzik. Whitepaper - semantic technologies - a comparison.

- [12] F. Wawrzik, W. Chipman, J. M. Molina, and C. Grimm. Modeling and simulation of cyber-physical systems with sicyphos. In *2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, 2015.
- [13] F. Wawrzik, J. M. Molina, and C. Grimm. A concept for design of embedded systems at semantic level. In *Proceedings of the 2014 Forum on Specification and Design Languages (FDL)*, volume 978-2-9530504-9-3, pages 1–4, 2014.

Remarks:

Publication 12 and 13 preceded the thesis topic and are listed for completeness. Item 3 and 6 were (invited) presentations, publication 7 and 11 are whitepapers.

Frank Wawrzik

Curriculum Vitae

"There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle." - Albert Einstein

Education

- Spring 2022 **Ph.D. Thesis "Knowledge Representation in Engineering 4.0"**, *TU Kaiserslautern*, Kaiserslautern.
- March 2014–present **Research Associate at chair of design of cyber-physical systems**, *TU Kaiserslautern*, Kaiserslautern.
- March 2014 **Master Thesis "Ontologies for Parameterizing Electric Building Blocks with Transaction Level Modeling"**, *Frank Wawrzik*, TU Kaiserslautern, Kaiserslautern.
- 2012 **Bachelor Thesis "Examination of Battery Management Systems"**, *Frank Wawrzik*, TU Kaiserslautern, Kaiserslautern.
- October 2006 **Study course electrical engineering with major in wireless communication at TU Kaiserslautern**, Kaiserslautern.

Research and Applied Experience

Teaching

- March 2014–2021 **Teaching Assistant at TU Kaiserslautern**, *Seminars, Master Thesis, Bachelor Thesis and Master Projects*.

Funded and Participating Projects

- 2018 **Arrowhead Tools**, *coordinated by Prof. Dr. Jerker Delsing*, Lulea University of Technology, EU Horizon 2020 project.
ARROWHEAD TOOLS FOR ENGINEERING OF DIGITALISATION SOLUTIONS
- 2018 **GENIAL!**, *coordinated by Dr. Berthold Hellenthal*, Audi AG, BMBF project.
COMMON ELECTRONICS ROADMAP FOR INNOVATIONS IN THE AUTOMOTIVE VALUE CHAIN
- 2016 **VICINITY**, *coordinated by Prof. Dr. Christoph Grimm*, TU Kaiserslautern, EU Horizon 2020 project.
FROM SILOS TO DECENTRALIZED, CROSS-DOMAIN/CROSS-VENDOR IoT INTEROPERABILITY
- 2015 **KvUiCPS**, *supervised by Prof. Dr. Christoph Grimm and Dirk Denger*, TU Kaiserslautern and AVL List GmbH, Graz.
COMPOSITIONALITY OF UNCERTAINTIES IN CYBER-PHYSICAL SYSTEMS

2014 **ANCONA**, supervised by Prof. Dr. Christoph Grimm, TU Kaiserslautern, BMBF project.

CROSS-LEVEL VERIFICATION OF MIXED SIGNAL CIRCUITS

Other Projects

2012 **German Research Center for Artificial Intelligence Internship**, supervised by Prof. Dr. Detlef Zühlke, Department of Smart Factory and Industry 4.0, Kaiserslautern.

EMBEDDED SYSTEMS PROGRAMMING IN C, SOA, RFID-READER COMMUNICATION

Technical Skills

Programming Languages Java, C,C++, Assembler

Technical Softwares MATLAB, Protégé (OWL), Papyrus, Enterprise Architect, etc (SysML), SystemC, Website Technologies, L^AT_EX, ArangoDB, MySQL, Spring, REST

Basic Utilities Microsoft Office, Git

Operating Systems MacOS, Windows

Language Skills

English	Advanced	<i>Conversationally fluent and fluent in writing (e.g. scientific papers)</i>
German	Advanced	<i>Conversationally fluent (mother tongue)</i>
French	Basic	<i>High school level French</i>

References

1. **Prof. Dr. Christoph Grimm**, Full Professor, TU Kaiserslautern, Kaiserslautern.
email : grimm@cs.uni-kl.de
2. **Prof. Dr. Raúl García-Castro**, Associate Professor, Universidad Politécnica de Madrid (UPM), Madrid.
email : rgarcia@fi.upm.es