Design Rationale bei der Modellierung und Abwicklung von Entwurfsprozessen

Kirstin Kohler

Diplomarbeit
Universität Kaiserslautern
September 1995

Betreuung:
Prof. Dr. Michael M. Richter
Dr. Frank Maurer

Erklärung

Hiermit versichere ich, daß ich diese Arbeit selbständig angefertigt und keine anderen Hilfsmittel, als die von mir angegebenen, verwendet habe.

Kaiserslautern, den 11. September 1995

Kirstin Kohler

Inhaltsverzeichnis

Kapitel 1

Einleitung und Überblick

1.1 Einleitung

Die Planung, Steuerung und Verwaltung komplexer Entwurfsprozesse wird durch die folgenden Charakteristika erschwert:

Lange Projektlaufzeiten: Der Entwurf umfangreicher Produkte erstreckt sich häufig über mehrere Jahre. In dieser Zeit werden sehr viele Einzelentscheidungen bezüglich des Entwurfes getroffen, die in ihrer Summe die Eigenschaften des fertigen Produktes festlegen.

Viele Mitarbeiter: Am Entwurf sind eine Vielzahl von Mitarbeitern beteiligt, von denen jeder durch seine Arbeit einen Teil des Produktes erstellt. Jeder einzelne Mitarbeiter trifft zu diesem Zweck Entscheidungen, die die Eigenschaften des Produktes bestimmen.

Implizites Wissen über komplexe Zusammenhänge: Entwurfsentscheidungen werden häufig erst nach einer Reihe von Überlegungen getroffen, die sehr vielfältige Aspekte berücksichtigen. Das Wissen, das den Entscheidungen zugrunde liegt, ist häufig nur implizit in den Köpfen der Entwickler vorhanden, die die Entscheidung treffen.

Da Entscheidungen auf komplexen Überlegungen basieren, die nicht explizit dargestellt werden, sind viele der Entscheidungen nicht nachvollziehbar. So kann sich der Entwickler, der die Entscheidung getroffen hat, häufig selbst nicht mehr erinnern, welche Überlegungen zu dieser Entscheidung geführt haben. Anderen Mitarbeitern ist ohne eine entsprechende Dokumentation ohnehin nur schwer möglich den Entwurf nachzuvollziehen. Lange Projektlaufzeiten erschweren den Prozeß zusätzlich, weil die Gründe für Entscheidungen nach einiger Zeit in Vergessenheit geraten. Infolgedessen führt die Kombination der oben genannten Eigenschaften dazu, daß Abhängigkeiten zwischen verschiedenen Entscheidungen mit fortschreitender Zeit und wechselnden

Mitarbeitern in Vergessenheit geraten, so daß eine Wiederverwendung oder Überarbeitung des bestehenden Entwurfes erheblich erschwert oder gar unmöglich wird.

Um diese Probleme zu handhaben, bietet sich eine computerunterstützte Planung und Abwicklung komplexer Entwurfsprozesse an. Der Computer soll die Dokumentation des impliziten Wissens ermöglichen und die Menge an Information konservieren. Er soll der Vielzahl von Mitarbeitern ein geeignetes Instrument zur Kommunikation zur Verfügung stellen, um sich über eigenen Entscheidungen und die anderer Mitarbeiter zu informieren. Außerdem soll er die komplexen Zusammenhänge zwischen Entscheidungen verwalten, um den Anwender durch geeignete Methoden den Umgang mit diesen Abhängigkeiten zu erleichtern

Im Rahmen von Arbeiten der AG Richter wurde solch ein System unter dem Namen CoMo-Kit [Mau93] entwickelt. Im Zusammenhang mit Arbeiten an CoMo-Kit wurden eine Reihe von Techniken entwickelt, die dazu beitragen, komplexe Entwurfsprozesse plan-, steuer,- und kontrollierbar zu machen, indem sie die oben beschriebene Unterstützung des Entwicklungsprozesses ermöglichen.

- Mit Hilfe eines Werkzeuges kann ein Modell des Entwurfsprozesses erstellt werden. Diese Modell beschreibt zum einen, in welche Teilaufgaben sich der Entwurfsprozess aufgliedert und welche Methode zur Bewältigung dieser Aufgaben verwendet werden können. Zum anderen kann beschrieben werden, welche Teilprodukte oder Informationen von den einzelnen Teilaufgaben verwendet werden und welche Produkte durch diese Aufgaben erzeugt werden.

- Desweiteren stellt CoMo-Kit eine Ausführungskomponente zur Verfügung, mit deren Hilfe die Teilaufgaben an verschiedene Mitarbeiter delegiert werden können. Diese Ausführungskomponente stellt sicher, daß nur die Aufgaben abgearbeitet werden können, für die auch alle notwendigen Informationen zur Verfügung stehen. Zusätzlich unterstützt sie Änderungen im Entwurf, indem sie Abhängigkeiten, die zwischen verschiedenen Teilen des Entwurfes bestehen verwaltet. Diese Abhängigkeiten werden aus der Information des oben beschriebenen Modelles generiert. Sie machen es möglich Veränderungen im Entwurf selektiv an die davon betroffenen Mitarbeiter zu propagieren.

  Diese Arbeit beschäftigt sich damit, wie die durch das Modell gegebenen Abhängigkeiten des Entwurfsprozesses noch verändert werden müssen, um den aktuellen Zusammenhang des Entwurfes genauer zu beschreiben. Gibt das Modell die Abhängigkeiten zwischen den Entscheidungen nicht vollständig wieder, so soll es möglich sein, die Abhängigkeiten entsprechend an zupassen. Je genauer die Zusammenhänge zwischen den Entscheidungen beschrieben werden, desto besser kann diese Information genutzt werden, um auf Veränderungen im Entwurf, wie oben beschrieben, zu reagieren.

Die Techniken werden in zwei Anwendungsdomänen geprüft und weiterentwickelt, in der Bebauungsplanung [MP95] und im Software Engineering. In der Bebauungsplanung assistiert CoMo-Kit den Planer bei der Erstellung eines Bebauungsplanes und erleichtert das Verständnis des fertigen Planes. Im Rahmen des Sonderforschungsbereiches 501 wird CoMo-Kit zur Planung und Abwicklung von Softwareentwurfsprozessen eingesetzt.

1.2 Aufgabenstellung

Diese Diplomarbeit beschäftigt sich mit der Erfassung und Verwaltung von Entwurfsentscheidungen und deren Begründungen. In der Literatur wurde in diesem Zusammenhang der Begriff Design Rationale geprägt (eine genaue Begriffsdefinition erfolgt im Abschnitt 2.1).

Begründungen von Entscheidungen spielen bei der Betrachtung von Entwurfsentscheidungen unter verschiedenen Gesichtspunkten eine sehr wichtige Rolle.

- **Darstellung:** Zum einen ermöglicht eine Darlegung der Begründungen das Verständnis der Entscheidung.
- **Auswertung der Entscheidung:** Zum anderen beschreibt die Begründung die Qualität der Entscheidung. Ist die Begründung der Entscheidung falsch, so muß die Entscheidung selbst in Frage gestellt werden.
- **Abhängigkeiten zwischen Entscheidungen:** Begründungen verdeutlichen häufig den Zusammenhang zwischen verschiedenen Entscheidungen. Kann dieser Zusammenhang dargestellt werden, so kann untersucht werden, welche anderen Entscheidungen betroffen sind, wenn eine existierende Entscheidung zurückgenommen werden muß

Nur unter Berücksichtigung dieser drei Aspekte, kann eine computerunterstützte Reaktion auf Änderungen im Entwurf möglich sein. Werden Entscheidungen und ihre Begründungen dargestellt, die Abhängigkeiten zwischen ihnen beschrieben und ist bekannt, wie die Gültigkeit der Argumente, den Zustand der Entscheidung bestimmt, kann eine Propagierung von Änderungen über dieser Darstellung die Auswirkung der Änderung verdeutlichen.

Für die Diplomarbeit ergaben sich aus dieser Thematik die folgenden drei Aufgaben:

- Eine ausführliche Literaturstudie soll einen Überblick über bestehende Techniken und Systeme darstellen, die sich mit der Thematik Design Rationale beschäftigen.
- Auf der Grundlage der Literaturstudie soll festgestellt werden, welche Anpassungen und Erweiterungen in CoMo-Kit notwendig sind, um Begründungen von Entwurfsentscheidungen darzustellen und zu verwalten.
- CoMo-Kit soll dann um die beschriebenen Erweiterungen durch eine entsprechende Implementierung ergänzt werden.

1.3 Übersicht

In Kapitel 2 wird der Begriff Design Rationale aus dem Blickwinkel verschiedener Ansätze definiert. Das Kapitel liefert einen Überblick über den aktuellen Stand der Forschung. In diesem Zusammenhang werden verschiedene Repräsentationsformen erläutert, mit deren Hilfe Entscheidungen und deren Begründungen dargestellt werden können.

In Kapitel 3 wird erläutert, welche Strukturen in CoMo-Kit zur Beschreibung von Begründungen und Entscheidungen und deren Verwaltung bestehen. Zu diesem Zweck wird eine Einführung in CoMo-Kit die grundlegenden Strukturen und Funktionen erklären, anschließend wird eine Zuordnung der Begriffe zu der Thematik Design Rationale vorgenommen.

Auf diesen Darstellungen aufbauend, werden in Kapitel 4 die Anpassungen analysiert, die in CoMo-Kit wünschenswert sind. Anschließend wird die Realisierung dieser Anpassungen dargestellt.

Im Kapitel 5 werden die Funktionen des erweiterten CoMo-Kit mit den Systemen aus Kapitel 2 verglichen.

Ein Beispiel soll in Kapitel 6 die beschriebenen Ergebnisse verdeutlichen.

Das letzte Kapitel stellt eine Zusammenfassung der Arbeit dar und gibt Hinweise auf mögliche Weiterentwicklungen.

Die Systemdokumentation und Schnittstellenbeschreibung befindet sich im Anhang.

Für das Verständnis der Kapitel 3 bis 4 sollte der Leser Grundkenntnisse im Bereich Planung und Truth Maintenance Systeme [Doy79] besitzen. Eine Lektüre der Arbeiten von Maurer [Mau93] und Dellen [Del94] zum Thema CoMo-Kit und der Arbeit von Petrie [Pet91] und Ritzer [Rit93] zum Thema REDUX ist keine Notwendigkeit, für tiefergehende Fragen und genaue Definitionen wird jedoch auf diese Arbeiten verwiesen.

Ein Teil der Ergebnisse dieser Arbeit wurden in Form eines Beitrages zur ICSE96 eingereicht (B.Dellen, K.Kohler und F.Maurer: Integrating Software Process Models and Design Rationales).

Kapitel 2

Design Rationale: Ein Überblick zu Nutzen und Anwendung

In der Literatur wurde der Begriff Design Rationale im Zusammenhang mit Entwurfsentscheidungen und deren zugrundeliegenden Überlegungen geprägt. Der Begriff **Design Rationale** hat in den letzten Jahren zunehmend Beachtung im Gebiet der Künstlichen Intelligenz und des Software Engineering gewonnen. In der Künstlichen Intelligenz wird Design Rationale im Zusammenhang mit der Entwicklung von wissensbasierten Systemen zur Unterstützung von Entwurfsprozessen verwendet [Kle94, FMM89, GR91, AID92] Im Bereich Software Engineering beschäftigt man sich mit DR beim Entwurf von Softwaresystemen [DJ88, PB88, JLS92, HA93, RD94].

Im Rahmen der Ausführungen sollen die verschieden Zielen, Repräsentationsformen und Fähigkeiten verschiedener Systeme aufgezeigt werden. Diese Beschreibung der einzelnen Ansätze wird ohne eine Wertung oder einen Vergleich der Arbeiten untereinander erfolgen. Auf diese Weise wird die Grundlage geschaffen, die Anforderungen im Bezug auf die Thematik DR für CoMo-Kit zu konkretisieren, um anschließend entsprechende Aspekte in CoMo-Kit integrieren zu können. Wie die Ausführungen dieses Kapitels die Arbeiten an CoMo-Kit beeinflußt haben werden die Kapitel 3 und 4 verdeutlichen.

DR wird in verschiedenen Arbeiten unter sehr unterschiedlichen Gesichtspunkten betrachtet. Die folgenden Erläuterungen sollen einen Überblick über die verschiedenen Auffassungen des Begriffs DR liefern. Zu diesem Zweck werden in Abschnitt 2.1 die einzelnen Begriffsdefinitionen erläutert. Anschließend werden in Abschnitt 2.2 die Ziele dargestellt, die einzelne Autoren mit der Verwendung von DR verfolgen. In einem weiteren Abschnitt werden dann die verschiedenen Repräsentationsformen erläutert, die sich aus den unterschiedlichen Motivationen ergeben. Einige Repräsentationsformen dienen der reinen Dokumentation für den menschlichen Anwender, andere bilden die Basis für eine computerunterstützte Verarbeitung des Wissens. Es werden einige konkrete Systeme vorgestellt, in denen DR den Entwurfsprozeß auf die eine oder andere Weise unterstützt.

Zur Beschreibung der einzelnen Repräsentationsformen werden die englischen Orginalbezeichnungen übernommen, um dem interessierten Leser die Einarbeitung in die Orginalliteratur zu erleichtern.

2.1 Begriffsbestimmung: Design Rationale

Es konnte in der Literatur keine allgemein anerkannte Bestimmung des Begriffes gefunden werden. Es ist vielmehr so, daß jeder Autor seinen Arbeiten eine eigene Begriffsbildung zugrunde legt. Die unterschiedliche Verwendung der Bezeichung **Design Rationale** soll durch die folgenden Erläuterungen dargelegt werden, um dem Leser die vielfältigen Variationen zu verdeutlichen

- In einer sehr allgemeinen Form versteht man unter DR die **Dokumentation** des Gedankenganges (*Rationale*), den ein Designer seinem Entwurfsprozeß zu Grunde legt.

- Thomas Gruber [GR92] präzisiert den Begriff DR wie folgt: DR ist eine Erklärung bezüglich jeder notwendigen Information über einen Entwurf. **Erklärung** ist in diesem Zusammenhang eine strukturierte Darlegung bestehend aus Begründungen für und gegen eine Position. Unter **notwendiger Information** versteht er Information, die notwendig ist, um eine bestimmt Aufgabe zu erfüllen.

  DR beinhaltet für Gruber sowohl Information über den Entwurfsprozeß, als auch über die Produkte des Entwurfs.

- Eine andere Definition von Gruber und Russel [GR91] bezieht sich auf den Begriff der Entwurfsgeschichte. Als Entwurfsgeschichte bezeichnen sie die Dokumentation der Entwurfsveränderungen in wiederverwendbarer Form. DR sind Antworten bezüglich der Produktbeschreibung, die aus Informationen der Entwurfsgeschichte generiert werden. Damit bezieht sich DR auf jede Art der Beschreibung, die während des Entwurfsprozessen generiert und modifiziert wird. DR beschränkt sich nicht nur auf Begründungen des Entwurfes.

- Yakemovic und Conklin [YC90, CY91] verstehen unter DR die historische Aufzeichnung der Gründe für die Wahl der Produkteigenschaften.

  Lee [LL91] schlägt zur Darstellung dieser historischen Aufzeichnung eine Strukturierung der Information vor. Diese wird durch explizite Darstellung der logischen Zusammenhänge (unterstützende Argumente für einen Vorschlag), und/oder durch Darstellung der historischen Zusammenhänge (ein Vorschlag ersetzt einen anderen Vorschlag) erreicht.

- Jarczayk et al. [JLS92] verstehen unter DR die explizite Aufzeichnung der Entscheidungen, die während des Entwurfsprozesses getroffen werden, sowie der Gründe dafür, daß genau diese Entscheidungen getroffen wurden. Dabei scheint es den Autoren sinnvoll, nur die Art von Entscheidungen zu betrachten, die Grundlage einer Diskussion sind. Damit macht Jarczayk deutlich, daß sich seine Sichtweise auf den Entscheidungsprozeß konzentriert.

Lee [Lee90, LL91] verwendet zur Präzisierung von Design Rationale den Begriff **Decision Rationale**. Er versteht darunter die Darstellung aller Überlegungen, die zu einer Entscheidung geführt haben.Dieses Verständnis vom Begriff DR deckt sich mit der Vorstellung von Jarczaryk.

- Caroll und Rosson [CR91] verstehen darunter die Aufzählung aller Behauptungen, die in den Entwurf des Produktes eingehen. Diese müssen wahr sein, wenn das Produkt erfolgreich verwendet werden kann. Zusätzlich beschreibt DR die physiologischen Folgen für den Benutzer des zu entwickelnden Produktes.
- MacLean et al. [MYBM91] definieren DR als Beschreibung des **Design spaces**. Sie verstehen unter *Design space* den Raum der möglichen Designalternativen, in dem das entwickelte Produkt eingeordnet ist.
- Pena und Logcher [PL92] definieren DR als die Sammlung aller Informationen, die durch die historische Entwurfsgeschichte, die Spezifikation der Charakteristika des Entwurfes und die Beschreibung der Pläne zur Erstellung des Produktes gegeben sind, mit besonderer Beachtung der Annahmen, die während des Entwurfes gemacht werden. Dieses Verständnis von DR kombiniert die oben genannten Vorstellungen von Gruber, Yakemovic, Jarczayk und Lee.

Welche Begriffsbestimmung in CoMo-Kit verwendet wird, soll in späteren Ausführungen deutlich werden.

2.2 Motivation für Design Rationale

Ein allgemeines Ziel von DR ist es, Information über den Entwurfsprozeß explizit zu erfassen, die häufig nur in Form impliziten Wissens in den Prozeß einfließt. Welche Art von Information zu diesem Zweck aufgezeichnet wird, folgt unmittelbar aus der Motivation des betreffenden Ansatzes.

Wie durch die Analyse der Begriffsdefinition bereits angedeutet, betrachten die Autoren DR mit unterschiedlichen Zielsetzungen.

Im folgenden sind alle wesentlichen Motivationen zusammenfassend aufgeführt, die die verschiedenen Autoren veranlassen, sich mit DR zu beschäftigen.

1. **Verdeutlichung von Anforderungen und Annahmen über das funktionsfähige Endprodukt des Entwurfsprozesses**. Der Designer kann sich verdeutlichen, von welchen Annahmen er implizit bei seinem Entwurf ausgegangen ist. Dadurch kann er Inkonsistenzen oder falsche Annahmen erkennen. Er kann Klarheit darüber gewinnen, an welcher Stelle noch ein Informationsbedarf besteht (z.B. bezüglich der Anforderungen des Kunden) [MYBM91].

2. **Unterstützung bei der Auswahl zwischen verschieden Alternativen:** Durch die Repräsentation der Argumentation kann der Designer verschiedene Entwurfslösungen erkennen und sie gegeneinander abwägen [MYBM91].

3. **Verständnis des bestehenden Entwurfes:** Dieser Aspekt ist unter verschiedenen Aspekten von Bedeutung:

   - Die Darstellung des Entwurfs oder Entwurfsprozesses kann die Kommunikation zwischen verschiedenen Teammitgliedern, die gemeinsam an einem Entwurf arbeiten, erleichtern [MYBM91]. Man spricht in diesem Zusammenhang von *Group Decision Support* [Zac86].
   - Bei Austausch einzelner Teammitglieder soll das Know-How nicht verloren gehen [PL92].
   - In verschiedenen Entwicklungsstadien des Produktes sollen die Teammitglieder in der Lage sein, zeitlich zurückliegende Entwurfsentscheidungen zu verstehen. Dies ist insbesondere im Zusammenhang mit Wartung und Wiederverwendung wichtig [HA93, DJ88, FMM89, MYBM91, ABCF91].

4. **Untersuchung der Wirkung von Veränderungen in Anforderungen oder Entwurf:** Dieses Wissen erleichtert die Überarbeitung und Wiederverwendung bestehender Entwürfe [Kle94, RD94].

5. **Erklärungen zum erwarteten Verhalten des zu entwerfenden Produktes:** Durch diese Erklärungen kann der Designer beim Entwurf die Eigenschaften des Produktes berücksichtigen, die das geforderte Verhalten zeigen [GR91].

6. **Verifikation und Validierung des Entwurfes:** Überprüfung, ob das Produkt den Anforderungen genügt [Kle94, RD94].

In Abhängigkeit von der Zielsetzung findet die durch DR explizit dargestellte Information Verwendung in verschiedenen Abschnitten des Entwurfsprozesses.

- beim konzeptuellen Entwurf (vgl. Punkt: 1, 2, 3, 5)
- bei der Fehlerbehebung (vgl. Punkt: 3, 6)
- bei der Wartung [DJ88] (vgl. Punkt: 3, 4)
- bei der Überarbeitung (vgl. Punkt: 3, 4)
- bei der Wiederverwendung und Modifikation bestehender Entwürfe [DJ88] (vgl. Punkt: 3, 4)

2.3 Repräsentationsformen von DR

Verfolgt der Autor ein konkretes Ziel mit der Dokumentation von DR, so hat dies nicht nur unmittelbare Folgen auf die Art der Information, die zu diesem Zweck repräsentiert wird, sondern auch auf deren Formalisierungsgrad und die Methoden und Werkzeuge mit denen auf diese Information wieder zugegriffen wird (bzw. auf mögliche Inferenzsysteme).

Die verschiedenen Repräsentationsformen für DR sind infolgedessen sehr vielfältig in ihrem Formalisierungsgrad. Sie reichen von Hypertextnetzen bis hin zu entscheidungstheoretischen Modellen

Die Repräsentationsformen von Abschnitt 2.3.1 bis 2.3.8 beziehen sich auf die Darstellung von Entscheidungen. Zur Darstellung von Entscheidungen muß man möglichen Alternativen, Argumente für und gegen die einzelnen Alternativen und Begründungen, die eine der Alternativen unterstützt, betrachten. Eine Begründung basiert auf Wissen über den Entwurf. Diese Art von Repräsentationsformen werden als **argumentationsbasierte Formen** bezeichnet. Argumentationsbasierte Systeme stellen Veränderungen im Laufe des Entwurfsprozesses als Entscheidungen dar und dokumentieren die Begründungen dieser Entscheidungen. Die verschieden argumentationsbasierten Notationen unterscheiden sich im wesentlichen durch die Konstrukte, die Argumente darstellen, durch die Beziehung zwischen den Konstrukten, sowie durch den Detaillierungsgrad, in dem Wissen dargestellt wird. Shum et al. [SH94] und Jarczyk et al. [JLS92] liefern einen umfassenden Überblick zu den argumentationsbasierten Repräsentationsschemata.

Im Unterschied zu argumentationsbasierten Repräsentationsformen basieren **modellorientierte Darstellungen** nicht auf einer Darstellung des Argumentationsraumes, sondern auf der Modellierung des zu entwerfenden Produktes. Durch die Modellierung kann das Verhalten des Entwurfsobjektes vorhergesagt bzw. geprüft werden. In Abschnitt 2.3.9 wird eine solche Repräsentationsform dargestellt.

Zur Erläuterung der Repräsentationsformen wird für jede Repräsentationsform zuerst die zugrundeliegende Sichtweise des Entwicklungsprozesses beschrieben. Anschließend werden die Darstellung und das Vokabular des Formalismus selbst erklärt und deren besondere Merkmale hervorgehoben.

Die Reihenfolge, in der die einzelnen Formalismen aufgeführt sind, gibt in etwa die chronologische Abfolge der Entwicklung wieder. Parallel zur zeitlichen Entwicklung ist eine zunehmende Strukturierung und Formalisierung der Repräsentationsformen festzustellen.

2.3.1 Toulmins Argumentationsschema

Dieses Schema ist die älteste Repräsentationsform zur Darstellung von DR-Systemen [Tou58]. Es wurde als Ergebnis philosophischer Studien des Argumentationsprozesses entwickelt.

Toulmins Argumentationsschema wird hier nur kurz wiedergegeben, um die

Vollständigkeit der historischen Entwicklung aufzuzeigen und die Wurzel der Entwicklungen auf dem Gebiet DR aufzuführen.

Toulmin hat die logische Struktur einer Argumentationskette identifiziert und in eine graphische Darstellung überführt. Die Notation besitzt fünf Komponenten, die über vier Relationen in einer Beziehung zueinander stehen:

- **Datum:** beschreibt eine Beobachtung.
- **Warrant:** gibt eine Rechtfertigung an.
- **Claim:** Auf Grund der Rechtfertigung *Warrant* wird eine Annahme *Claim* über den Gegenstand der Argumentation *Datum* gemacht.
- **Backing:** unterstützt die Gültigkeit der Rechtfertigung *Warrant*.
- **Rebuttal:** beschreibt eine Ausnahme von der Regel, die durch Regel aus *Datum*, *Warrant* und *Claim* beschreiben wird.

In Abb. 2.1 sind die Komponenten und ihre Relationen anhand eines Beispiels verdeutlicht.

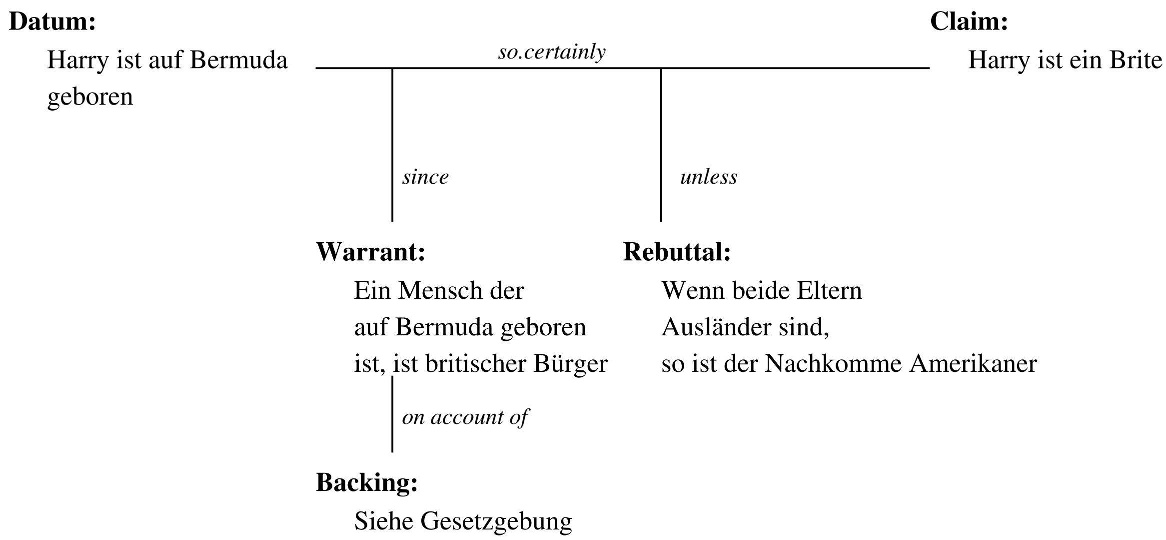

Abbildung 2.1: Ein Beispiel zur graphischen Argumentationsstruktur nach Toulmin

Eine große Schwachstelle dieses Systems liegt darin, daß die Zusammenhänge zwischen einzelnen *Claims* nicht dargestellt werden können.

2.3.2 Issue-Based-Information-System (IBIS)

Der Formalismus wurde von Kunz und Rittel [KR70] entwickelt und unter dem Namen IBIS veröffentlicht (IBIS steht für **I**ssue-**B**ased-**I**nformation-**S**ystem). Eine ganze Reihe von Arbeiten haben auf dieser Formalisierung aufgebaut und IBIS in Form verschiedener Dialekte weiterentwickelt.

Annahmen des Formalismus

Der Entwurfsprozeß wird als Ergebnis aller Überlegungen und Konversationen aufgefaßt, die Entwerfer, Kunden und Benutzer in den Planungsprozeß einbringen. Jeder Teilnehmer einer Diskussion bringt seine Argumente und Sichtweisen ein, die dazu beitragen sollen, ein Problem zu lösen. Die Modellvorstellung geht davon aus, daß die Diskussion des Entwurfes alle Probleme spezifiziert.

Beschreibung der Repräsentationsform

Alle Hauptanliegen, die Gegenstand einer Diskussion sind, werden als *Issue* bezeichnet. Auf jedes Issue wird durch eine oder mehrere *Positions* eingegangen. *Arguments* unterstützen (*supports*) oder widersprechen (*object to*) den einzelnen Positionen. Die Beziehungen zwischen Issues selbst können durch verschiedene Relationen (*generalize, spezialize, replaces, questions oder is-suggested-by*) dargestellt werden.

IBIS besitzt somit 3 verschiedene Knotentypen *(Issues, Positions und Arguments)* und 8 verschiedene Kantentypen *(supports, object-to, replaces, responds-to, generalizes, specializes, questions und is-suggested-by).*

Die Grundstrukturen einer IBIS-Darstellung ist in Fig. 2.2 abgebildet.

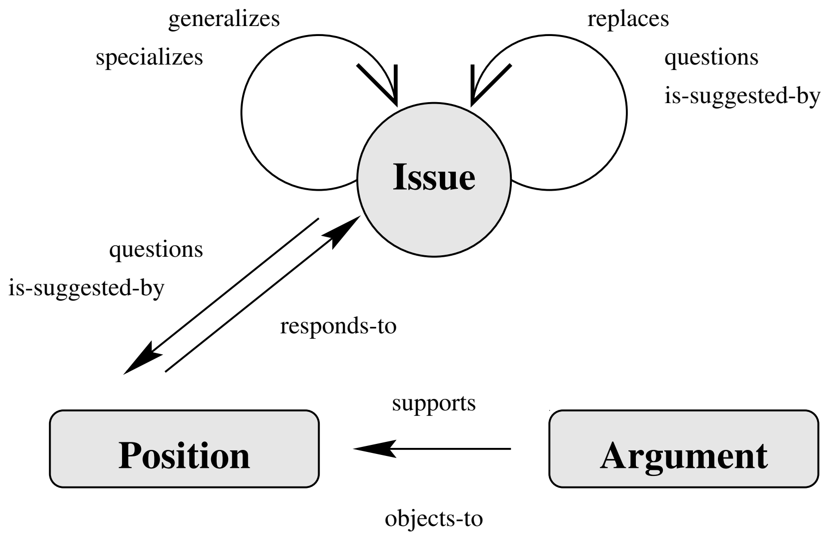

Abbildung 2.2: Grundstrukturen eines IBIS-Darstellung

Folgende verschiedenen Darstellungsformen bauen auf dem IBIS-Formalismus auf.

it-IBIS steht für **i**ntendet-**t**ext-IBIS. It-IBIS verdeutlicht hierarchische Abhängigkeiten von Knoten durch Einrückung des Textes. Issues werden mit **I**, Positions mit **P**, unterstützende Argumente mit **AS** und ablehnende Argumente mit **AO** markiert. Gelöste Issues und ausgewählte Positions werden mit einem Stern ($\star$) gekennzeichnet. Ist noch keine Entscheidung getroffen, so wird dies durch ein Fragezeichen (?) dargestellt. Wurde eine Position zurückgewiesen so ist dies am Minuszeichen (**-**) zu erkennnen. In Tabelle 2.1 ist eine Beispiel dieser Darstellungsform aufgeführt. Zur einer tieferen Studie dieser Notation wird auf die Literatur [YC90] verwiesen.

g-IBIS ist ein graphisches Hypertext-Softwaretool zum Aufbau von IBIS-Netzen. Die Issues, Positionen und Argumente selbst werden in informeller Form dargestellt und bilden durch die oben beschriebenen Relationen (supports, objects to, ...) ein semi-strukturiertes Hypertextnetzwerk. Zur Ansicht eines Bildschirmabzug aus einer gIBIS-Struktur wird auf die Orginalliteratur verwiesen (siehe: [CY91], Abb.12, S.364; [YC90], Abb.3, S.110).

Aus den Arbeiten zu gIBIS ist nach Angaben von Shum und Hammond [SH94] das einzige kommerziell vertriebenen DR-Tool (CMSI,1992 [Sys]) hervorgegangen.

PHI steht für **P**rocedural **H**ierarchy of **I**ssues. PHI erweitert den IBIS-Ansatz, in dem es eine erweiterte Definition des Begriffes Issue zuläßt und die Relationen zwischen Issues variiert. Issues sind in PHI nicht nur Fragen zu Designentscheidungen, die Gegenstand der Überlegungen sind, sondern beliebige Designentscheidungen. Die Beziehungen zwischen Issues werden auf die Relation *serve* eingeschränkt. Diese Relation ist wie folgt definiert:

Issue A *serves* issue B, wenn die Antwort auf A die Antwort zu Issue B bestimmt.

Die Darstellung der Argumentation mittels einer PHI-Struktur ergibt durch die Beschränkung auf die Relation *serve* eine quasi-hierarchische Struktur [McC86]. Nach Ansicht der Autoren liegt ein Vorteil von PHI darin, jeder Aufgabe und Teilaufgabe des Entwurfsprozesses ein Issue und Subissue zuzuordnen. Die Wurzel einer PHI-Struktur repräsentiert das Ziel des gesamten Entwurfsprozeßes.Eine ausführliche Beschreibung zu PHI und dessen Anwendung befindet sich in [FMM89, FLMM91].

Merkmale der Repräsentationsform

IBIS ist ein rethorisches Modell zur Darstellung der Dialoge und Diskussionen, die zum Entwurf eines Produktes führen. Dabei spiegelt IBIS eine natürliche Art und Weise wieder, Entscheidungen darzulegen.

$\star$**I:** Welcher Prozessor soll gewählt werden ?
  ?P: Prozessor A
    AS: schnell
  $\star$**P:** Prozessor B
    AS: schon in Gebrauch, also billiger
  -P: Prozessor C
    AO: Ist zur Zeit nicht lieferbar
      $\star$**I:** Kann er bald geliefert werden ?
        $\star$**P:** Nein
          AS: Das Design ist erst nächstes Jahr fertig

Tabelle 2.1: Beispiel einer it-IBIS Diskussion

IBIS kann, im Gegensatz zu Toulmins Argumentationsschema, Abhängigkeiten zwischen verschiedenen Behauptungen aufzeigen.

Conklin und Jakemovic [CY91] betonen, daß IBIS sich insbesondere eignet, um Entwicklungsprozesse zu unterstützen, die kaum standardisiert sind und deren Lösungstechnologien weniger bekannt sind. Die Darstellung des DR wird als prozeß-orientiert bezeichnet, da DR, abgefaßt in IBIS, eher die Geschichte des Entwurfsprozesses wiedergibt als die verschiedenen Möglichkeiten des Produktentwurfes strukturiert darzustellen. Conklin und Jakemovik nennen IBIS Darstellungen aus diesem Grund deskriptive Beschreibungen des Entwurfsprozesses. IBIS eignet sich also eher als Dokumentationswerkzeug , das den Zustand eines speziellen Entwurfes darstellt, als zur Wiederverwendung in generischen Prozeßmodellen.

Die IBIS-Darstellung hat den Vorteil, daß die Dokumentation relativ einfach ist. Der Benutzer muß keine formalen Sprachen erlernen. Ein entscheidender Nachteil liegt allerdings darin, daß die informelle Repräsentation die computerunterstützte Auswertung des dargestellten Wissens kaum zuläßt. Insbesondere erlaubt IBIS keine quantitative Entscheidungsunterstützung (decision support). Es ist außerdem nicht möglich, in IBIS Constraints auszudrücken, die sich zwischen verschiedenen Issues ergeben. Wird zum Beispiel für Issue I1 die Position P1.1 ausgewählt und macht diese Auswahl die Position P2.1 eines anderen Issues I2 ungültig, weil sich die beiden Positionen gegenseitig ausschließen, so kann diese gegenseitige Einschränkung der Issues in IBIS nicht dargestellt werden.

Computerunterstützung auf einer IBIS-Darstellung ist in folgenden Bereichen möglich:

Views: Die Darstellung des IBIS-Netzes kann auf die offenen Issues beschränkt werden, die zusätzlich nach absteigender Priorität sortiert sind.

Syntax-Check: Der Benutzer kann gewarnt werden, wenn er gegen einfache Syntaxregeln der IBIS-Struktur verstößt (z.B.: Ein Issue ist als gelöst markiert, obwohl keine Position ausgewählt wurde).

Entscheidungsunterstützung: In g-IBIS besitzen *Issues* und *Arguments* das Attribut *Importance* (mögliche Wertebelegung: high, medium, low). Beträgt der Wert dieses Attributes für einen Knoten entweder high oder low, so wird der Knoten gekennzeichnet, damit der Benutzer den Knoten besonders beachtet.

Lubars hat eine Version von g-IBIS entwickelt, in der der Glaubenszustand automatisch durch das TMS-Netzwerk propagiert wird. In Abschnitt 2.3.8 wird dieses System näher beschreiben.

Weitere Mechanismen zur Unterstützung durch das Gesamtsystem sind in der Arbeit von Conklin und Yakemovic [CY91] aufgeführt.

2.3.3 REMAP

REMAP (**Re**presentation and **Ma**intenance of **P**rocess Knowledge) [RD94] unterstützt die Entwicklung und Wartung von großen Softwaresystemen. Zu diesem Zweck stellt das System verschiedene Werkzeuge zur Verfügung, mit deren Hilfe das zu entwickelnde Produkt und die Elemente der Argumentation beschrieben werden. Diese Darstellung ermöglicht den Entwurf mit Hilfe der Inferenzmechanismen des Systems auszuwerten.

Die Autoren von REMAP gehen davon aus, daß ein Entwurf von einer informell beschriebenen Benutzeranforderung ausgeht, die die zu erfüllenden Ziele des Entwurfsprozesses repräsentiert. Der Entwurf beinhaltet die Verfeinerung, Modifikation und Ausarbeitung der Anforderungen. Die Erfüllung der Ziele des Entwurfes wird durch Entwurfsentscheidungen erreicht, die die Lösung mittels Constraints zunehmend einschränken.

Beschreibung der Repräsentationsform

REMAP verwendet eine Darstellungsform, die auf den IBIS-Strukturen (vgl. Abschnitt 2.3.2) aufbaut (*Issue, Position, Argument)*. Allerdings wird die IBIS-Notation um folgende Elemente erweitert:

- **Requirements:** Anforderungen, an das zu entwickelnde System.
- **Assumptions:** Annahmen, die den Argumenten zugrunde gelegt werden
- **Criteria:** Beurteilung, die verwendet werden, um die *Issues* zu lösen.
- **Design Decisions:** Entscheidungen, die beschreiben, wie *Issues* gelöst werden. Diese Entscheidungen beruhen auf der Wahl einer *Position*.

- **Constraints und Design Objects:** *Decisions* resultieren in einer Eingrenzung der möglichen Produkteigenschaften. Diese Eingrenzungen werden durch *Constraints* beschrieben. Die *Constrainsts* legen die Produkteigenschaften fest, definieren also letztendlich die *Design Objects*.

Die folgenden Beziehungen beschreiben die Zusammenhänge der einzelnen Elemente:

Beziehungen zwischen Arguments and Assumptions: Eine oder mehrere *Assumptions* beschreiben die Rechtfertigung eines *Arguments*. Das *Argument* wird als gültig bezeichnet, wenn die *Assumptions* der Rechtfertigung einen gültigen Glaubenszustand besitzen. Das *Argument* kann mehrere Rechtfertigungen besitzen, die sich jeweils wiederum aus mehreren *Assumptions* zusammen setzen.

Beziehungen zwischen Positions and Arguments: Eine *Position* kann in Erwägung gezogen werden, wenn alle *Arguments*, die sie unterstützen gültig sind, und keines der *Arguments*, die sie widerlegen gültig ist.

Beziehungen zwischen Decisions und Positions: Eine *Decisions* ist gültig, wenn die gewählte *Position* gültig ist.

Beziehungen zwischen Constraints und Decisions: *Constraints* werden durch *Decisions* gestützt, die sie hervorrufen.

Beziehungen zwischen Objects und Constraints: Die Gültigkeit von *Design Objects* wird durch das Bestehen oder Fehlen von *Constraints* bestimmt, die die *Objects* auswählen oder sie von der Lösung des Entwurfes ausschließen.

Generalisierungshierarchie: *Anforderungen, Issues* und *Decisions* werden jeweils durch ein Generalisierungshierarchie beschrieben. Diese unterstützt eine rekursive Verfeinerung des Entwurfes durch die Zerlegung von Zielen und Teilziele.

Die beschriebenen Elemente der Repräsentationsform und die Beziehungen zwischen diesem Elementen ist in Abbildung 2.3 dargestellt. Die IBIS-Strukturen sind durch die gestrichelte Linie gekennzeichnet. Diese Abbildung wurde aus der Orginalliteratur (vgl. [RD94], Abb.1) übernommen.

Die beschriebenen Beziehungen der Strukturen werden in REMAP mittels eines *Reason Maintenance Modules* verwaltet. Es ist aus der Literatur leider nicht ersichtlich, mit Hilfe welcher Technik die Abhängigkeiten in diesem Modul verwaltet werden, um die Auswertung der Beziehungen nach der oben dargestellten Semantik zu unterstützten.

Merkmale des Systems

REMAP erweitert die Darstellungsform von IBIS, in dem zusätzlich beschrieben werden kann, wie Issues gelöst werden, auf welchen Annahmen die Argumente beruhen

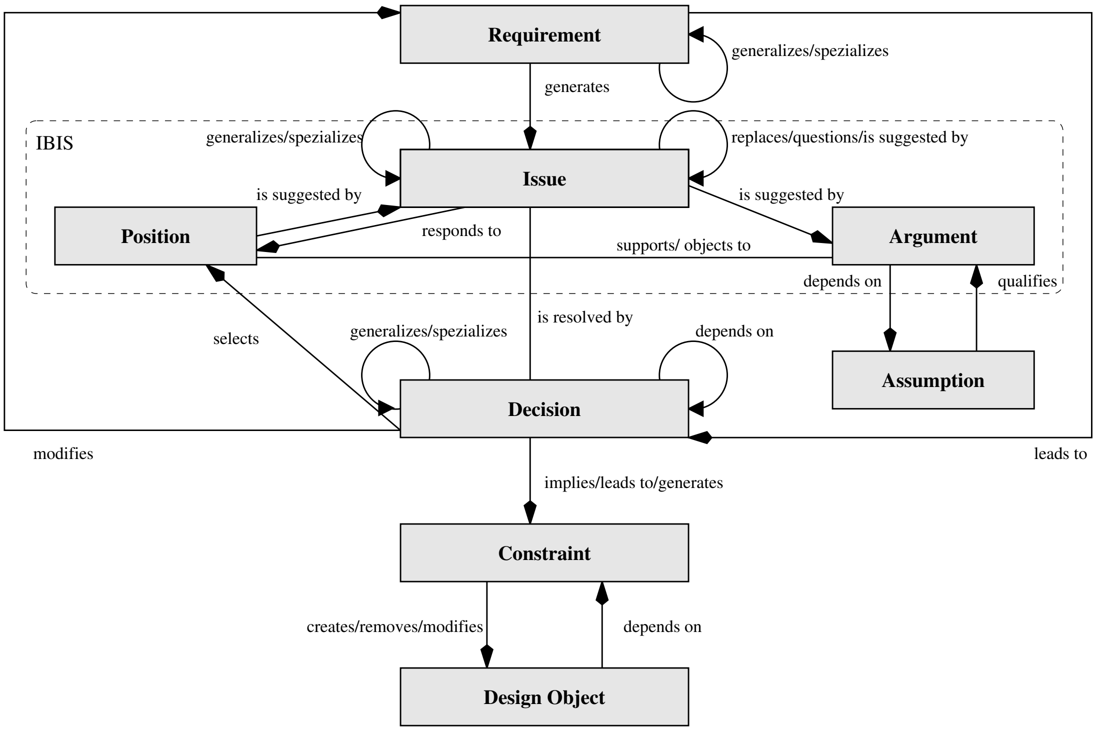

Abbildung 2.3: Basisstrukturen und Beziehungen in REMAP

und welche Bewertungen herangezogen werden, um die Issues zu lösen. Außerdem ist es in REMAP möglich, die Ergebnisse des Entwurfsprozesses zu den Elementen der Argumentation in Relation zu setzten.

Indem REMAP eine Darstellung der Entwurfsgeschichte unterstützt, wird das Verständnis des Entwurfes erleichtert. Durch die Verwaltung der Abhängigkeiten mittels eines *Reason Maintenance Module* können Veränderungen von *Assumptions, Decision* oder *Requirements* über die gesamte Repräsentation des Entwurfs propagiert werden. Dabei ist es von besonderem Interesse, wie sich dies auf die *Constraints* auswirkt. Auf die beschriebene Weise ist REMAP in der Lage Veränderungen von Entwurfsentscheidungen auf Änderungen in Entwurfslösungen weiterzupropagieren.

Der Formalismus unterstützt einen evolutionären Designprozeß, in dem REMAP die Schleifen der Iteration beschreibt, die zwischen Requirements und Entscheidungen bestehen. Diese kommen zu stande, wenn die Entscheidungen, die getroffen werden um Anforderungen zu erfüllen, wiederum zur Modifikation und Verfeinerung von Anforderungen führen.

2.3.4 Question-Option-Criteria (QOC)

MacLean, Young, Belloti et al. [MYBM91, Bel93, MM94] repräsentieren DR durch **"Design Space Analyse"**. Der Design Space stellt das entwickelte (zu entwickelnde) Produkt in einem Raum verschiedener Alternativen dar und macht deutlich, warum das Produkt aus den verschiedenen Alternativen gewählt wurde. MacLean et al. haben zur Darstellung eine semiformale Notation entwickelt, die als **Q**uestion-**O**ptions-**C**riteria (QOC) bezeichnet wird.

Beschreibung der Repräsentationsform

Die Abbildung 2.4 zeigt ein Beispiel der QOC-Repräsentation, deren Elemente in den folgenden Ausführungen beschrieben werden. Die Erläuterungen und das Beispiel sind der Orginalliteratur entnommen [MYBM91]

Der Formalismus besitzt drei Strukturen (Questions, Options und Criteria), die folgende Funktion besitzen:

Questions: sind Fragen, die wichtige Designeigenschaften betreffen. Die Fragen haben generativ und strukturierende Aufgaben. Sie strukturieren *Options*, indem sie zwischen verschiedenen *Options* einen Zusammenhang aufstellen. Außerdem werden mit ihrer Hilfe verschiedene *Options* generiert.

Options: sind mögliche Antworten auf *Questions*.

Criteria: dienen dazu, *Options* zu vergleichen und zu bewerten. Aus diesem Grund wird ein *Criterium* auf alle *Options* einer Frage angewendet.

Als viertes Element des Formalismus können zusätzlich *Arguments* aufgeführt werden, die erläutern sollen, inwiefern ein *Criteria* eine *Option* unterstützt oder widerlegt.

Questions, Options, Criteria und *Arguments* stellen die Knoten eines Graphen dar, dessen Kanten den Zusammenhang zwischen den vier Elementen aufzeigen. Folgende Kantentypen werden unterschieden:

Kanten zwischen Questions und Options können angeben, auf welchen *Question* die *Option* sich bezieht, das heißt, für welche Eigenschaft die *Option* eine mögliche Belegung liefert. Es ist auch möglich, daß eine *Option* eine weitere Frage aufwirft. Dieser Zusammenhang wird ebenfalls durch eine Kante dargestellt. Faßt man die Kanten als gerichtete Kanten auf, so besitzen die beiden beschriebenen Kantentypen entgegengesetzte Richtung.

Kanten zwischen Options und Criteria können eine positive (unterstützende) oder negative (ablehnende) Bedeutung besitzen, in Abhängigkeit davon, ob das *Criterium* die *Option* unterstützt oder nicht. Eine weitere Abstufung in der Beurteilung des Zusammenhanges (z.B.: stark positiv, positiv, mittel positiv, neutral, ...) halten die Autoren für zu komplex und unübersichtlich. Die positive bzw. negative Bewertung verschiedener *Criteria* gegen eine*Option* hat relativen Charakter. Das Beispiel in Abb. 2.4 macht dies deutlich: Einen breiten Scrollbalken kann man leichter (positive Kanten) mit der Maus treffen als einen schmalen (negative Kante).

Kanten zwischen Criteria entstehen, wenn eine Eigenschaft durch eine andere Eigenschaft begründet werden kann. Das eine *Criteria* stellt in diesem Fall eine Spezialisierung des anderen *Criteria* dar. Das generalisierte *Criteria* kann als eine Art der Begründung für das spezielle *Criteria* angesehen werden. Im Beispiel von Fig. 2.4 ist *leicht mit der Maus zu treffen* eine Spezialisierung von *schnelle Benutzeraktion* und *wenig Benutzerfehler*

Kanten zu Arguments werden von den Kanten zwischen *Options* und *Criteria* zu den *Arguments* gezogen. Sie verstärken die Bedeutung des *Criteria* für die Gültigkeit der *Option*.

Die beschriebene Darstellung mittels Knoten und Kanten liefert eine Strukturierung des Design Spaces. Die Knoteninhalte werden informell dargestellt und lassen beliebige textuellen Belegungen zu.

Merkmale der QOC Darstellung

Die Autoren betonen, daß die QOC-Darstellung eines Entwurfs nicht als Aufzeichnung der Entwurfsgeschichte zu verstehen ist, sondern als Coprodukt während des Entwurfsprozesses entwickelt wird. Die QOC-Darstellung kann als Zusammenfassung

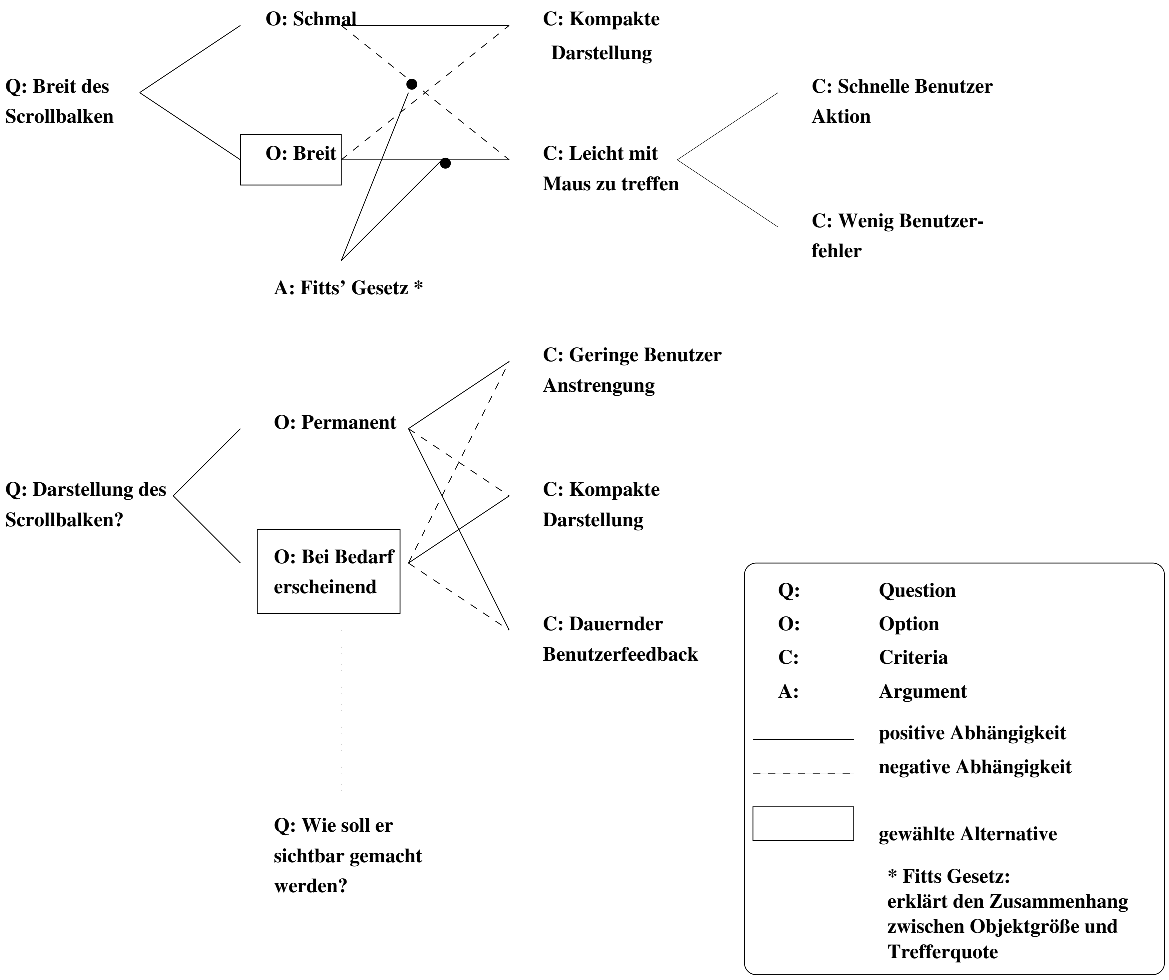

Abbildung 2.4: Beispiel einer QOC Darstellung: Design Space Analyse für die Entwicklung eines Scrollbalken für ein Fenstersystem

einer IBIS-Repräsentation aufgefaßt werden, die die wichtigsten Elemente als Grundlage einer logischen Argumentation in Zusammenhang bringt.

Aus diesem Grund eignet sich QOC im Gegensatz zur IBIS, um gut verstandene Entwicklungsprozesse zu unterstützen. Die Darstellung eignet sich besonders gut zur Wiederverwendung. Die QOC-Darstellung wird als strukturorientiert bezeichnet, weil DR als Repräsentation des Design Spaces verstanden wird.

Die Autoren lassen in ihren Ausführungen völlig offen, auf welche Art und Weise die Design Space Analyse erstellt werden soll und wie die Repräsentation als Grundlage eines computerunterstützten Entwicklungsprozesses verwendet werden kann.

2.3.5 Decision Representation Language (DRL)

Diese Repräsentationssprache wurde von Lee [Lee90, LL91] entwickelt. Die Sprache basiert auf einer Modellvorstellung, die die Information des Entwurfes in je einen Raum für Argumente, Alternativen, Auswertung der Alternativen, Kriterien und Ergebnisse (Sachverhalte) unterteilt.

Die 5 Räume gliedern die Information wie folgt:

- **Der Argumentations-Raum** enthält alle Argumente, die für den Entwurf eines Produktes von Bedeutung sind.
- **Der Alternativen-Raum** stellt alle möglichen Alternativen dar.
- **Der Auswertungs-Raum** Die Bewertung der einzelnen Alternativen wird in diesem Raum dargestellt.
- **Der Kriterien-Raum** beschreibt die Bewertungsmerkmale, die die Grundlage der Auswertung darstellen.
- **Der Ergebnis (Issue)-Raum** stellt den Zusammenhang zwischen verschiedenen Entscheidungen her.

Beschreibung des Repräsentationsformalismus

DRL besitzt ein festes Vokabular aus Objekten und Relationen, mit deren Hilfe die beschriebenen Räume dargestellt werden können. In Abbildung 2.5 sind die wichtigsten Objekte und Relationen der Sprache dargestellt. Diese Darstellung enthält keine vollständige Übersicht des Vokabulars.

Ihre Bedeutung wird in den folgenden Ausführungen beschrieben:

- **Objekte:**
  - **Alternativs:** stellen die Möglichkeiten dar, zwischen denen man wählen kann

  - **Goals** stellen die Eigenschaften dar, die die beste Alternativen besitzen sollte.
  - **Claims** beschreiben unsicheres Wissen in Form von Vermutungen oder Behauptungen.
  - **Decision-Problems** repräsentiert das Problem, das eine Entscheidung erfordert.

- **Relationen** sind immer Unterklassen von *Claims* (Behauptungen). In Abbildung 2.5 sind hinter den Relationen in runden Klammern die Objekte aufgeführt zwischen denen die Relation besteht *(Bsp: achieves(alternative,goal))*.

  Jeder *Claim* besitzt die folgenden drei Attribute.

  - **Evaluation** ist eine Funktion aus den beiden anderen Attributen *Plausibility* and *Degree*.
  - **Plausibility** gibt die Wahrscheinlichkeit für das Zutreffen des *Claims* an.
  - **Degree** gibt an, in welchem Ausmaß die Behauptung wahr ist (Bsp: *Degree* der Relation *achieves(alternative,goal)* gibt an, in welchem Maße die Alternative das *Goal* erreicht.

In Fig. 2.6 ist das Beispiel eines Entscheidungsgraphes in DRL dargestellt.
Die DRL-Elemente dienen dazu, die beschriebenen fünf Räume darzustellen:

Der Argument-Raum wird durch eine Menge von *Claims* dargestellt, die in einer Beziehung zueinander stehen. Mögliche Beziehungen zwischen *Claims* sind *supported, denied* und *presupposed*. In DRL können sich Claims direkt auf andere Claims beziehen.

Der Alternativen-Raum wird durch die *Alternatives* und ihre Relationen dargestellt. (Bsp: Die *is-a-kind-of*-Relation spezialisiert *Alternatives*.) Er ordnet den *Alternatives* die entsprechenden Attribute zu.

Der Auswertungs-Raum wird durch die Attribute *Evaluation, Plausibilität* und *Degree* zusammengesetzt. Jeder *Alternative* wird dadurch ein Wert zugeordnet, der ihre Relation zu den anderen *Alternatives* angibt. Der Wert der Gesamtauswertung wird durch die Auswertung des Attributes des *Claim is-a-good-alternative-for* zwischen Alternative und Auswertungsproblem dargestellt. Dieser Wert wiederum ergibt sich aus der Auswertung der *achieves*-Relation, die die *Alternatives* mit einem *Subgoal* des *Decision-Problems* verbindet usw.

Der Kriterien-Raum wird durch die *Goals* dargestellt, die wiederum die Bewertungsgrundlage der Auswertung beschreiben. Das *Decision-Problem* stellt das Ziel dar, die beste Alternative zu finden.

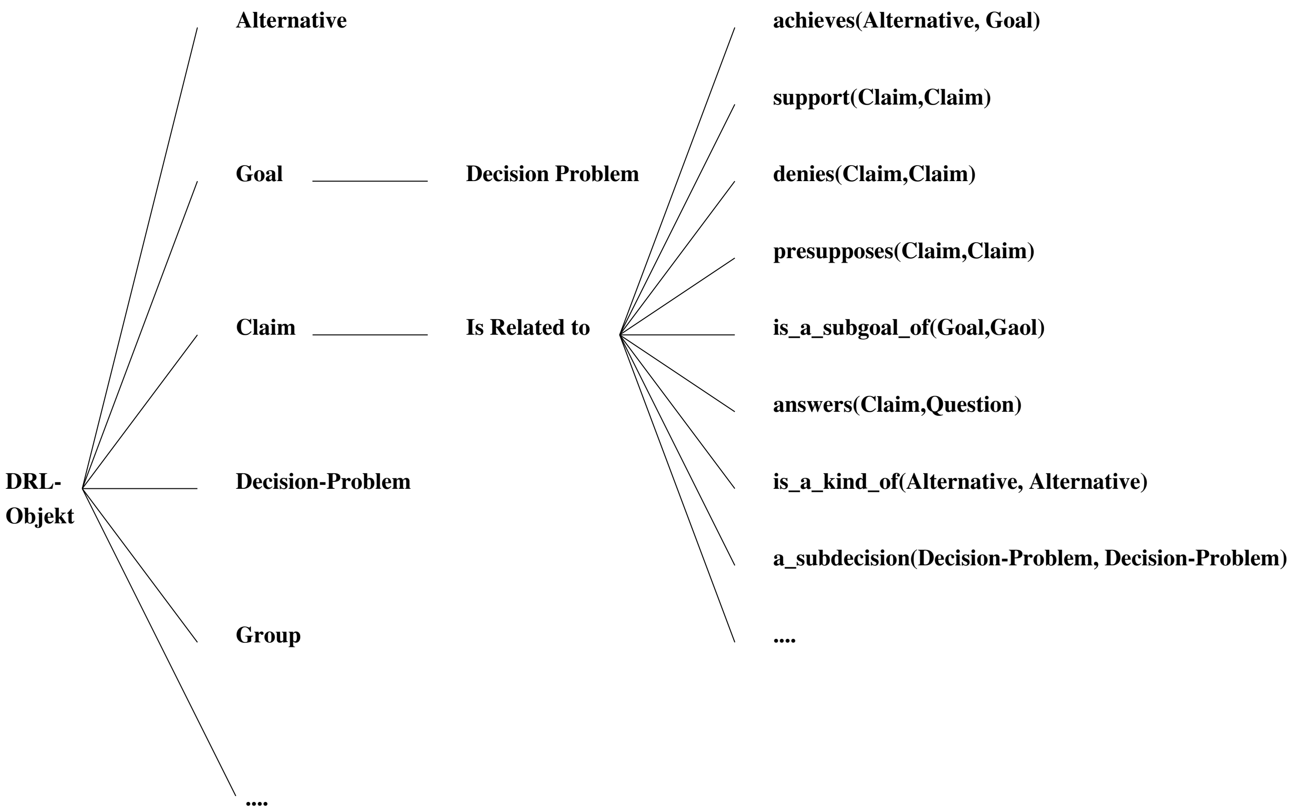

Abbildung 2.5: DRL Vokabular (Quelle: [LL91], Fig.3)

Der Issue-Raum wird durch die *Decision-Problems* beschrieben. Ein *Decision-Problem* kann durch eine *a-subdecision*-Relation mit einem anderen *Decision-Problem* in Beziehung stehen. Eine Einheit dieses Raumes besteht aus einer Entscheidung, die wiederum aus Alternativen-, Kriterium- und Auswertungsraum besteht.

Die 5 Räume sind folgendermaßen miteinander verbunden:

Verbindung zwischen Argumentations- und Alternativen-Raum: Für jede der *Alternative* gibt es eine Reihe von *Arguments*, die die gegenwärtige Alternativenauswahl beschreiben. Dabei ist es möglich, daß verschiedene *Alternatives* sich die gleichen *Arguments* teilen (Bsp: das Argument unterstützt eine Alternative und widerlegt eine andere).

Verbindung zwischen Argumentations- und Auswertungs-Raum: Die Verbindung zwischen *Argumentes* und *Alternatives* macht deutlich, warum eine *Alternative* den aktuellen Auswertungszustand besitzt.

Verbindung zwischen Argumentations- und Kriterien-Raum: Durch diese

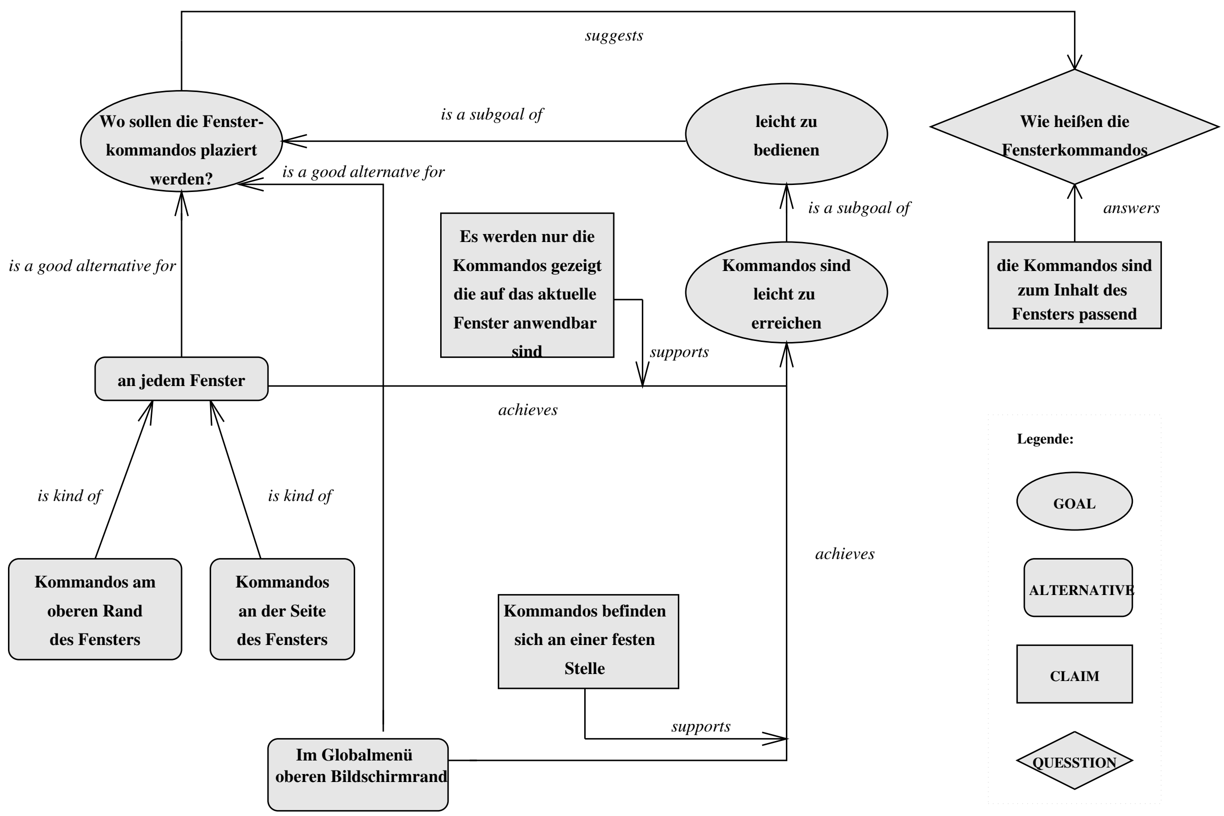

Abbildung 2.6: Beispiel für einen Entscheidungsgraphen nach Lee unter Verwendung von DRL (Auszüge aus [LL91], Abb.5)

Beziehung wird dargestellt, welche *Arguments* sich auf welche Bewertungsgrundlage beziehen.

Die Räume enthalten die Information, die notwendig ist, um mit Hilfe von DR eine Reihe von Fragen beantworten zu können. Die folgende Tabelle soll einen Überblick vermitteln, welche Räume dazu beitragen, welche Fragen zu beantworten.

Raum	Frage
Argumentations-Raum	Was wurde letzte Woche diskutiert?
Alternativen-Raum	Was sind die verschieden Entwürfe?
	Was sind die Vor- und Nachteile der Alternativen?
Auswertungs-Raum	Was sind die bisher besten Alternativen?
	Ändert diese Tatsache die Auswertung der Alternativen?
Kriterien-Raum	Muß diese Eigenschaften beibehalten werden?
	Warum ist diese Eigenschaft wichtig?
Ergebnis-Raum	Welche Probleme des Entwurfes sind ungelöst?

Abbildung 2.7 zeigt das beschriebene Modell mit seinen fünf Ebenen in Anlehnung an ein in [PL92] dargestelltes Beispiel.

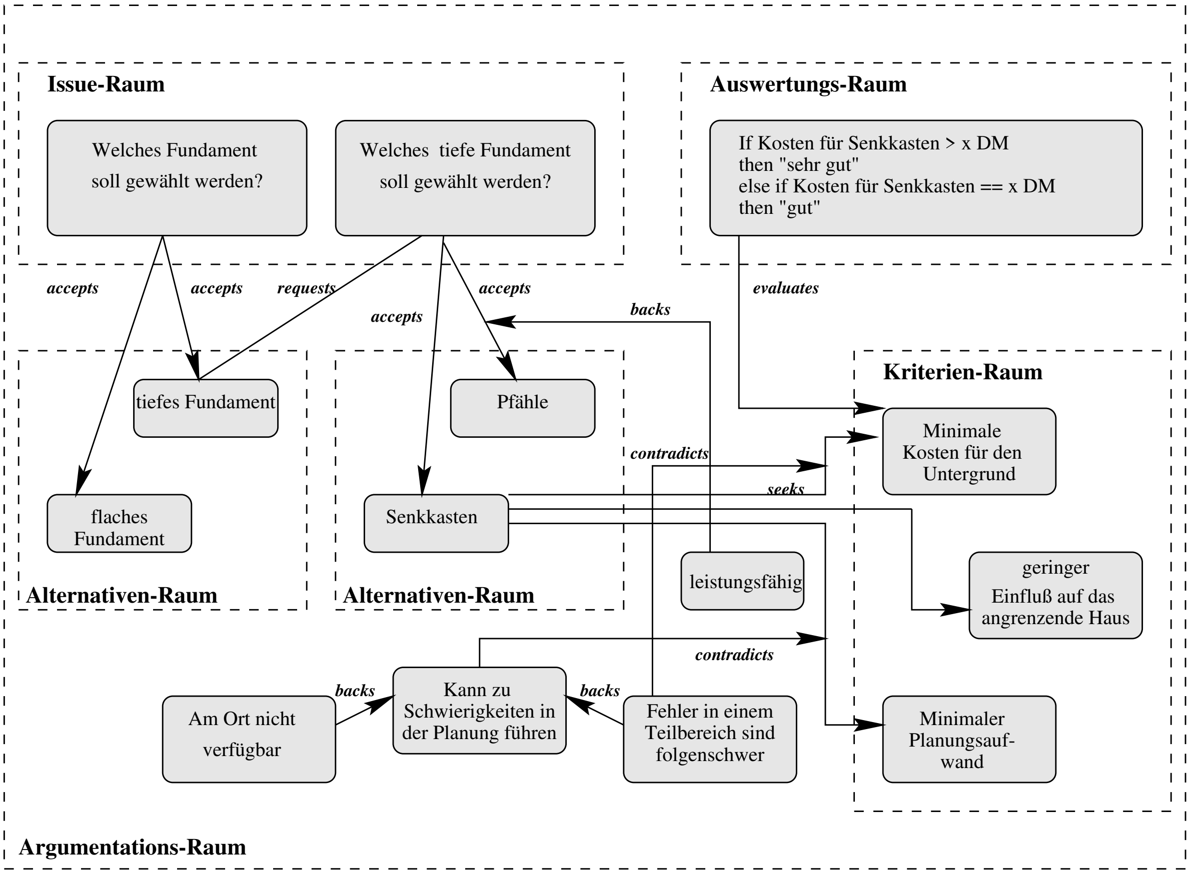

Abbildung 2.7: DR-Modell zur Wahl eines geeigneten Fundamentes beim Bau eines Hauses

Merkmale der Repräsentationsform

DRL ist sehr stark an die IBIS-Struktur angelehnt. Betrachtet man die Objekte und Relationen, die DRL zur Verfügung stellt, so finden sich die IBIS-Elemente in DRL unter anderer Bezeichnung wieder. Die *Decision-Problems* in DRL können mit den *Issues* von IBIS verglichen werden, die *Alternatives* mit den *Positions*, die *Claims* (denies und supports) mit den *Arguments*. Lee und Lai [LL91] selbst bezeichnen DRL als eine Erweiterung von gIBIS in folgenden Punkten:

Explizite Repräsentation den Kriterien-Raumes: IBIS sieht im Gegensatz zu DRL keine Darstellung von *Criterias* vor, so daß auch kein Beziehungen zwischen ihnen beschrieben werden können. Werden *Criterias* explizit dargestellt, so läßt sich deren exklusiver Ausschluß beschreiben. Wenn sich *Criterias* ändern, kann man die Auswirkungen dieser Änderung einfacher verfolgen.

Deutlichere Repräsentation des Argumentations-Raumes: *Arguments* können durch die Attribute quantifiziert werden. Alle Relationen werden als *Claims* dargestellt, so daß man auch über die Relationen selbst Vermutungen anstellen kann (Bsp: man glaubt an A und B, aber nicht an A *supports* B).

Infrastruktur zur Definition von Auswertungen: Durch die Definition der Attribute und die beschriebenen Abhängigkeiten sind die Voraussetzungen zur Auswertung in DRL geschaffen. Die Autoren schlagen jedoch kein spezielles Auswertungsverfahren vor. Sie weisen darauf hin, daß bei der Wahl einer bestimmten Bewertungsmethode auch entsprechende Algorithmen entwickelt werden müssen, die die Bewertungen über mehrere Ebenen von Vermutungen propagieren.

2.3.6 Repräsentationssprache von DRCS

DRCS steht für **D**esign **R**ationale **C**apture **S**ystem, einem System, das den Entwurf und die Darstellung von beliebigen Produkten unterstützt [Kle93] [Kle94]. Alle für den Entwurf notwendigen Informationen werden mit Hilfe einer eigenen Repräsentationssprache formalisiert.

Annahmen der Repräsentationsform

Produkte des Entwurfs werden als Sammlung von *Modules* interpretiert, aus denen sich das gesamte System zusammensetzt. Jedes *Modul* besitzt *Attributes*, die seine Eigenschaften beschreiben, und eine Schnittstelle. Über die Schnittstelle können die *Submodules* verbunden werden, um das Gesamtprodukt zusammenzusetzen. Die Schnittstelle wird ebenfalls durch *Attributs* beschrieben.

Pläne zum Entwurf werden als zeitlich angeordnete Abfolge von Aufgaben (*Tasks*) betrachtet. Die Aufgaben besitzen ebenfalls *Attributs*. Jede Aufgabe besteht aus einer oder mehreren atomaren Aktionen, die die Aufgabe erfüllen. Diese Pläne können sich auf die Generierung des Produktes beziehen (domain-level) oder sie können den Entwurfsprozeß selbst beschreiben (meta-level).

Beispiel:

Produktbeschreibung	zu entwerfendes Produkt:	Computer
	Module:	VLSI-Chips
	Modul-Attribute:	Leistung, Funktionalität
	Modul-Schnittstellen:	Pins
	Verbindung der Module:	Kabel
Plan(domain-level)	Aufgabe	Chips auf Board löten
Plan(meta-level)	Aufgabe	Anforderungen definieren Alternativen entwickeln

Pläne und Produkte werden in einem iterativen Prozeß verfeinert. Parallel zur Verfeinerung wird überprüft, inwieweit der Entwurf die Spezifikation erfüllt. Die Spezifikation des Entwurfes, die Ausarbeitung verschiedener Entwurfsalternativen, deren Auswertung oder Modifikation können sich während des Entwurfes beliebig überlagern. Dabei sind alle Überlegungen aus denen der Entwurf hervorgeht zielgerichtet (goal driven). Die Aktionen dienen immer zur Erfüllung eines der folgenden Ziele: Spezifikation erfüllen, Konflikte lösen, Verfeinerung des Entwurfes.

Beschreibung des Formalismus

Die Sprache besitzt zwei Hauptelemente: **Entities** (Objekte) und **Claims** (Vermutungen über die Objekte).

Entities: sind *Modules, Tasks, Spezifications* und *Versions.*

Claims: existieren in zwei verschiedenen Formen.

- **Relation Claims:** Vordefiniertes Vokabular zur Beschreibung von Beziehungen. (Beispiel: module-1 *has-submodule* module-2)
- **Text-Claims:** Beliebige textuelle Beschreibung, um Informationen darzustellen, die mit den vordefinierten Sprachkonstrukten nicht ausgedrückt werden können.

Mit Hilfe der beschriebenen Elemente läßt sich DR und Entwurf in Form einen Graphen darstellen, in dem Instanzen von *Entities* und *Text-Claims* durch *Relation-Claims* verbunden werden.

Das Vokabular kann in 5 Gebiete aufgeteilt werden, mit deren Hilfe jeweils einen Teil der Zusammenhänge dargestellt werden kann.

Beschreibung von Plan und Produkt: Die *Entities* zur Beschreibung von Produkten sind *Modules, Attributs, Interfaces* und *Connections. Modules* können aus *Submodules* oder *Spezifications* bestehen.

Die *Entities* zur Beschreibung von Plänen sind *Tasks*. Jeder Plan besteht aus einer Dekomposition von *Subtasks*, denen Aktionen zugeordnet sind, die die *Tasks* erfüllen.

Produktbeschreibung und Pläne zur Erstellung eines Produktes stehen über das Relation-Claim *"has-plan"* miteinander in Verbindung.

Attributs werden mit Hilfe einer Constraint-Sprache beschrieben.

Eine Übersicht zu Objekten und Beziehungen zu der Beschreibung von Plan und Produkt liefert Abbildung 2.8. In Abbildung 2.9 ist ein Beispiel zu den in Abbildung 2.8 dargestellten *Entities* und *Relations* dargestellt.

Hintergrund der Entscheidung: Die Definition von *Entities* und *Claims* verlangt vom Designer eine Entscheidung, diese wird durch ein *Decision-Problem* dargestellt.

(Beispiel: Die Relation *has-submodule* verlangt die Definition der *Submodules*. Das Decision-Problem ergibt sich aus der Frage nach der Dekomposition des Modules).

In Abb. 2.8 und 2.9 sind die entsprechenden Relationen abgebildet. Dort wirft die Beschränkung auf einen Radius von weniger als 180 feet die Frage *(Decision Problem)* auf, wie soll dies realisiert werden *(achieved by ?)*.

Versionen: Der Designer entwickelt immer dann neu *Versions*, wenn er eine Entscheidung fällt oder eine Alternative wählt.

Jede Alternative für ein *Decision-Problem* wird als eigene *Version* repräsentiert.

Die beste Lösung eines *Decision-Problems* wird durch die Zuordnung der Version über die *is-the-best-option-for* Relation dargestellt.

Eine Übersicht zum Vokabular von Objekten und Beziehungen liefert Abbildung 2.10.

Auswertung des Entwurfes: Die Spezifikation von Produkt und Plan sollen mit den Werten der Attribute von Produkt und Plan übereinstimmen. Plan und Produkt werden durch eine bestimmte *Version* repräsentiert.

Die Auswertung ist durch die Überprüfung der Relationen *achieves* zwischen *Spezifikation* und *Version* möglich.

Eine Übersicht zum Vokabular von Objekten und Beziehungen liefert Abbildung 2.10.

Argumentation: Die Elemente zur Beschreibung der Argumentation sind *Relation-* und *Text-Claims*, sowie *Procedures* und *Questions*.

Procedures können mathematische Gleichungen oder weniger strukturierte Information, wie textuelle Beschreibungen sein.

Questions können die Bewertung eines *Claims* in Frage stellten. Umgekehrt kann eine Frage durch einen *Claim* beantwortet werden. Als *Claim* kann hier jedes der oben beschriebenen *Claims* auftreten.

Eine Übersicht zum Vokabular von Objekten und Beziehungen ist in Abbildung 2.10 dargestellt.

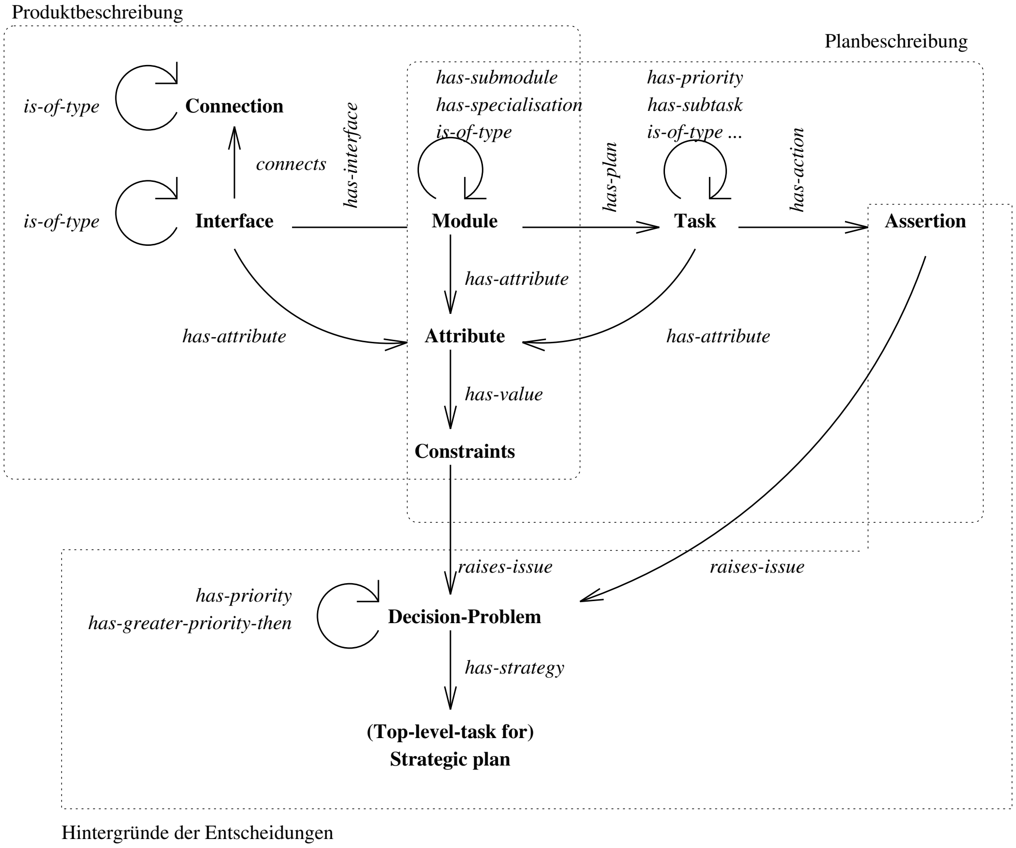

Abbildung 2.8: Vokabular zur Beschreibung von Plan und Produkt

Merkmale der Repräsentationsform

DRCS-Repräsentationssprache unterscheidet sich von den anderen bisher dargestellten Formalismen insofern, daß sie sich nicht zur Darstellung beliebiger Entscheidungsprozesse eignet, sondern sich speziell auf **Entwurfs**entscheidungen bezieht. Außerdem besitzt die Sprache ein Vokabular mit dem sowohl der Entscheidungsprozeß als auch das zu entwerfende Produkt und der Plan zur Generierung des Produktes dargestellt werden kann. Auf diese Weise besitzt die Sprache alle Voraussetzungen, um Abhängigkeiten zwischen Produkt, Plan und Entscheidungen zu modellieren. Der Formalismus erlaubt somit die Integration der Darstellungen von DR und Entwurf.

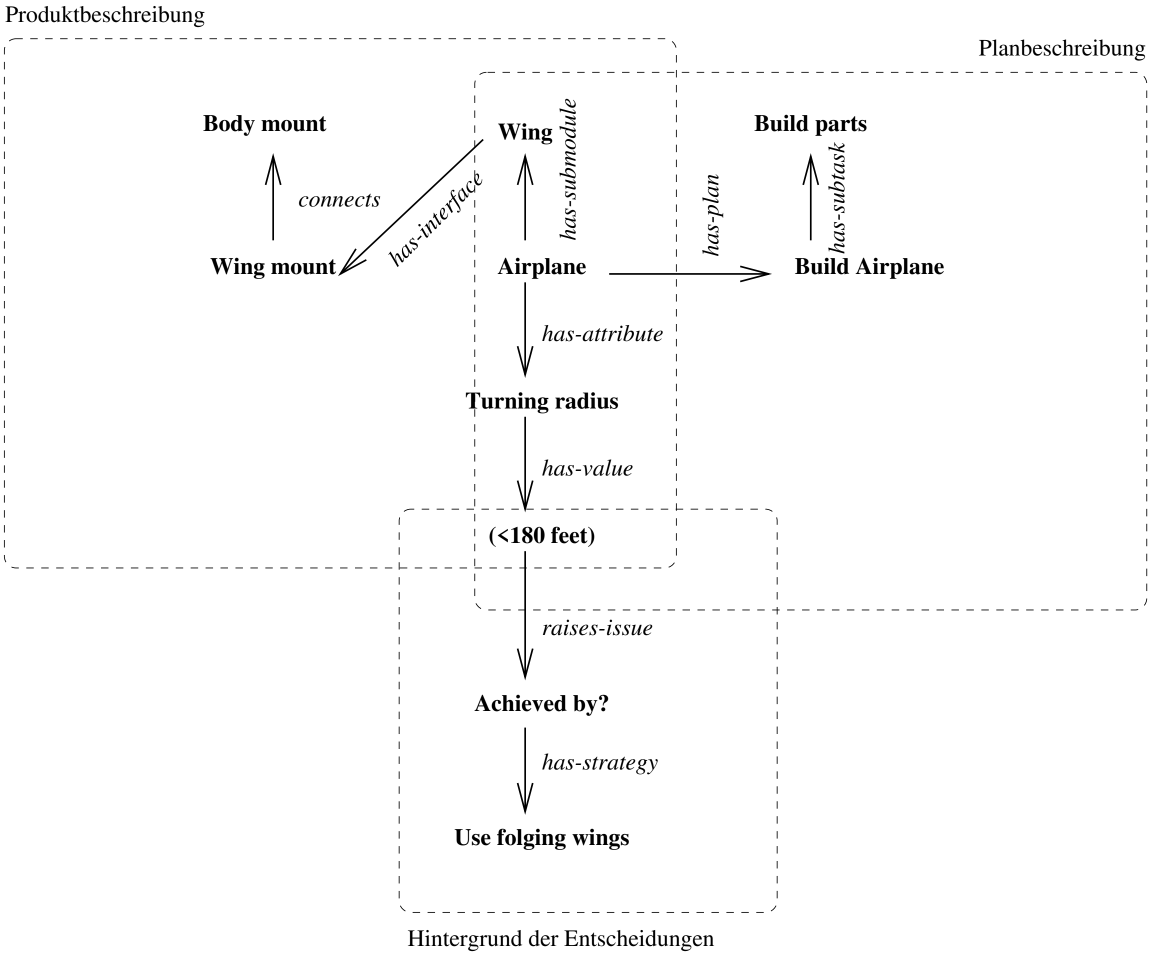

Abbildung 2.9: Ausschnitt aus einer Beschreibung von Plan, Produkt und Hintergrund. Die Anordnung entsprechend Abb.2.8 erleichtert die Zuordnung von Instanzen und Entities.

2.3.7 REDUX

REDUX ist ein von Petrie [Pet91, PCP94] entwickeltes System zum Planen und Umplanen (Replanning).

Sichtweise des Planungsprozesses

REDUX geht von der Vorstellung aus, daß Pläne inkrementell erstellt und ausgeführt werden. Infolgedessen sind Umplanungen notwendig, noch bevor der Plan vollständig erstellt oder ausgeführt ist. Die Planung beginnt mit einem Startziel, das in Teilziele verfeinert wird. Dieser Prozeß setzt sich rekursiv fort, bis auf einer sehr detaillierten Ebene die einzelnen Teillösungen gefunden werden, die in ihrer Gesamtheit die Lösung repräsentieren. Im Verlauf des verzahnten Planungs- und Ausführungspro-

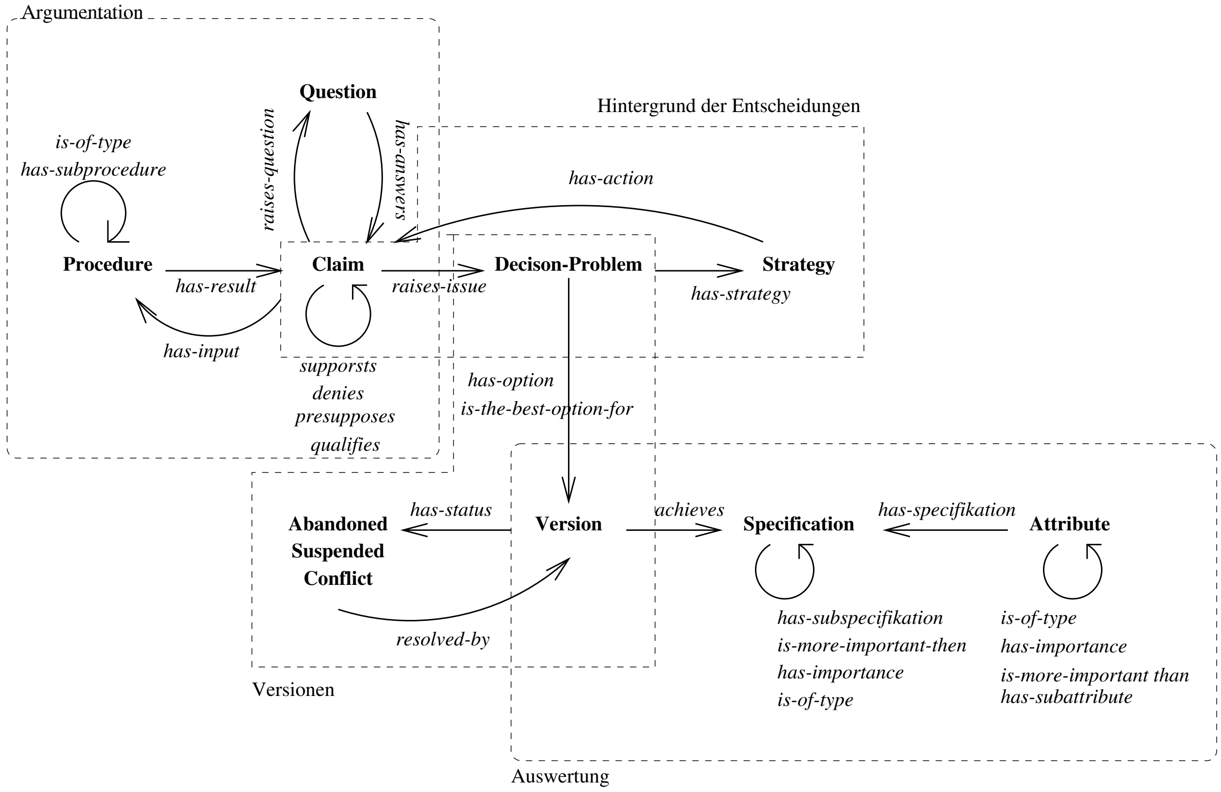

Abbildung 2.10: Vokabular zur Beschreibung von Argumentation, Version, Hintergrund und Auswertung des Entwurfes.

zesses ergibt sich häufig die Notwendigkeit der Umplanung. In REDUX können diese Umplanungen effizient behandelt werden, indem der existierende Plan entsprechend den neuen Anforderungen verändert werden kann. Der fertige Plan und das zu erstellende Produkt sind dann das Ergebnis der Verflechtung von schrittweisem Planen und Umplanen. REDUX unterstützt diesen Prozeß, durch eine Technik, die die Propagierung von Änderungen ermöglicht.

Repräsentationsform von REDUX

Folgende Begriffe werden in REDUX zur Beschreibung des Entwurfsprozesses verwendet:

- **Ziele:** werden durch Operatoranwendungen erreicht. In Abbildung 2.11 beschreibt die *"Wahl eines Motors"* das Ziel Z1. Jedem Ziel ist eine Menge von Operatoren zugeordnet, die zur Erfüllung des Zieles angewendet werden können, man bezeichnet diese Menge als *Konfliktmenge*.

- **Operatoren** beschreiben eine Zerlegung des Zieles in Teilziele und\oder bestimmen Variablenzuweisungen. In Abbildung 2.11 wird durch die Operatoranwendung ein neues Teilziel Z2 bestimmt und eine Variablenzuweisung *Motor = motor-1* vorgenommen.

- **Variablenzuweisungen** beschreiben die konkrete Erstellung eines Produktes oder Teilproduktes durch die Anwendung eines Operators.

- **Entscheidungen** stellen die Anwendung eines Operators zur Erreichung eines Zieles dar. Eine Entscheidung erzeugt Variablenbelegungen oder neue Ziele, je nachdem, ob es sich um einen atomaren oder komplexen Operator handelt. Die folgenden Eigenschaften bestimmen den Zustand der Entscheidung:

  – **Die Konsistenz** einer Entscheidung wird durch Bedingungen beschrieben, die erfüllt sein müssen, wenn die Entscheidung gültig ist. Für unser Beispiel könnte solch eine Bedingung die Lieferbarkeit von *motor-1* sein.

  – **Die Optimalität** einer Entscheidung wird durch das *Decision Rationale* dargestellt. Diese Bedingungen beschreiben, warum ein Operator aus seiner Konfliktmenge gewählt wurde. Im obigen Beispiel wurde *motor-1* dem *motor-2* vorgezogen, weil er billiger war.

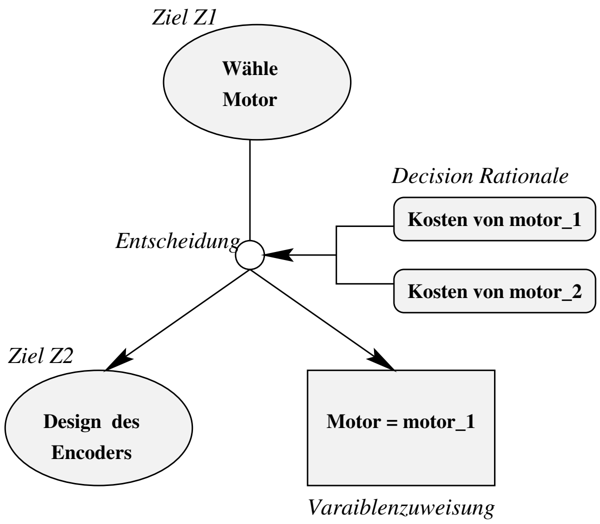

Abbildung 2.11: Darstellung eines Zieles und dessen Bearbeitung durch eine Operatoranwendung in REDUX (Quelle: Abb. 1,[PCP94])

In REDUX werden Abhängigkeiten zwischen den oben beschriebenen Elementen aufgebaut und für die Konsistenzhaltung und zur Unterstützung der Umplanung ausgenutzt. Es soll hier nicht detailliert beschrieben werden, wie diese Abhängigkeitsverwaltung realisiert wird. Der interessierte Leser sollte dies in der Orginalliteratur [Pet91, PCP94] nachlesen. Allerdings sollen die folgenden Erläuterungen verdeutlichen, auf welche Weise die Abhängigkeiten den Planungsprozeß unterstützten.

Konsistenzverletzung: Wird eine der Konsistenzbedingungen des Operators verletzt, so wird die entsprechende Entscheidung invalidiert, daß heißt, sie gehört nicht mehr zum aktuellen Lösungsweg. Die Ungültigkeit der Entscheidung wird auf die Variablenzuweisungen und Teilziele weiterpropagiert, die durch die zugehörige Operatoranwendung entstanden sind. Ist in unserem Beispiel der *motor-1* nicht lieferbar, so wird die Entscheidung zurückgezogen. Demzufolge gehört das Teilziel Z2 nicht mehr zum Plan und die Variablenzuweisung *Motor = motor-1* wird ungültig.

Die Wirkung auf den Zustand des Planes kann durch die Propagierung der Konsistenzverletzung sehr tiefgreifend sein. Wenn der Plan beim Auftreten der Konsistenzverletzung schon weit fortgeschritten ist, so kann es sein, daß sehr viel des bisherigen Lösungsweges infolge der Propagierung ungültig wird.

Optimalitätsverlust: Wird das *Decision Rationale* ungültig, so wird die entsprechende Entscheidung nicht invalidiert. Es handelt sich um einen Optimalitätsverlust. Der Verlust der Optimalität ist nicht mit einer automatischen Invalidierung verbunden. Der Benutzer wird an dieser Stelle nur darauf aufmerksam gemacht, damit er die Entscheidung selbst zurückziehen kann, wenn er es als nötig erachtet. Das System hält die bisherige Lösung aufrecht, reagiert also wesentlich schwächer. Der Anwender kann selbst entscheiden, ob die fehlende Optimalität Grund genug ist, den bestehenden Teil der Lösung zu verwerfen.

Wird in unserem Beispiel der Preis von *motor-1* erhöht, so daß dieser nicht mehr billiger ist als *motor-2*, so verliert die Entscheidung ihre Optimalität. Dennoch kann es sinnvoll sein, den entsprechende Lösungsweg aufrechtzuerhalten. Wenn zum Beispiel, die Mehrkosten von *motor-1* die Kosten einer Umplanung übersteigen würden.

Merkmale des Systems

REDUX konzentriert sich nicht allein auf die Darstellung der Argumentation, wie viele der vorne beschriebenen Systeme, sondern es stellt Zusammenhänge zwischen dem Produkt und dem Entwurfsentscheidungen eines Designprozesses dar. Diese Zusammenhänge werden in Abhängigkeiten der oben beschriebenen Elemente des Systems übertragen. Diese können dann zur Umplanung verwendet werden, um festzustellen, bis zu welchem Teil des Planes, die bisherige Lösung beibehalten werden kann, und an welchen Stellen Veränderungen vorgenommen werden müssen. Die Abhängigkeitsverwaltung unterstützt somit die Propagierung von Änderungen im Entwurf. Sie werden zu diesem Zweck mittels eines TMS verwaltet. Mit Hilfe des TMS wird die Konsistenz der Lösung aufrecht erhalten. Im Unterschied zu bisherigen Ansätzen, die auf einem TMS aufbauten, wurden in REDUX Methoden entwickelt, die eine Umplanung unterstützen, indem auch die Behandlung der aufgedeckten Inkonsistenzen berücksichtigt wird. REDUX unterstützt die Wiederherstellung von Konsistenz in

Folge einer aufgetretenen Inkonsistenz. Das Abhängigkeitsnetzwerk in REDUX reagiert, bedingt durch die oben beschriebene Unterscheidung der Begriffe Konsistenz und Optimalität einer Entscheidung, weniger aggressiv auf Veränderungen als reine TMS-Systeme.

2.3.8 Ein issuebasiertes TMS

Von Lubas [Lub91] wurde ein Ansatz entwickelt, der IBIS-Elemente in einem TMS darstellt. Durch die Kombination sollen die Vorteile beider Formalismen vereinigt werden. Die folgenden Ausführungen fassen die Ergebnisse der Arbeit von Lubas in Kürze zusammen.

DR wird in Form einer issuebasierten Struktur dargestellt, die durch informell repräsentierte Information erweitert werden kann. Das System besitzt die Fähigkeit zur Inferenz durch das zugrunde liegende TMS. Das TMS erlaubt dem Anwender zu prüfen, wie sich eine andere Lösung des Entwurfproblems auswirken würde. Die Propagierung von Abhängigkeiten wird in einer graphischen Darstellung in IBIS-Form verdeutlicht.

Die Autoren haben aus folgenden Gründen eine Kombination von IBIS und TMS gewählt:

- Ein TMS stellt die Abhängigkeiten deutlicher dar als ein Issue-based System, da ein großer Teil der Information des IBIS in informeller Form in den Argumenten enthalten ist.

- Dagegen stellt das IBIS die Zusammenhänge in einer übersichtlicheren Form dar, da es weniger Knoten und Kanten besitzt.

- Zu dem spiegelt die Information in Form von natürlicher Sprache eher den Gedankengang des Designers wieder als ein TMS.

- Die IBIS-Darstellung verdeutlicht dem Designer unmittelbar, welche Alternativen zur Wahl stehen. Und die Wahl zwischen verschiedenen Alternativen, wird nicht nur durch die Argumente für die Alternative getragen, sondern auch durch deren Gegenargumente.

Zur Verbindung der jeweiligen Vorteile wird die ursprüngliche TMS-Struktur folgendermaßen variiert:

- Annahmen, die das gleiche Topic betreffen, werden gruppiert. Dieser Gruppe von Annahmen wird eine Issue zugeordnet, das die Frage nach der richtigen Annahme innerhalb der gegebenen Gruppierung stellt.

- Annahmen und ihre Negationen werden zu einer einzigen IBIS-Position zusammengefaßt. Und Begründungen, die die negierte Form unterstützen, werden als Argumente aufgefaßt, die die Position widerlegen.

Dieser Zusammenhang wird dann durch veränderte Abhängigkeiten im TMS beschrieben. Es ergeben sich die folgenden Beziehungen, um eine Begründung im TMS zu darzustellen:

- **Assertion IN:** Die Position muß wahr sein, um das Argument zu unterstützen.
- **Assertion OUT:** Die Position muß falsch oder unbekannt sein, um das Argument zu unterstützen.
- **Negation IN:** Die Position muß falsch sein, um das Argument zu unterstützen.
- **Negation OUT:** Die Position muß wahr oder unbekannt sein, um das Argument zu unterstützen.

Durch diese Darstellung erhält man vier mögliche Glaubenszustände einer Position.

- (Assertion IN, Negation OUT) - die Position in wahr;
- (Assertion OUT,Negation IN) - die Position in falsch;
- (Assertion IN, Negation IN) - die Position hat einen widersprüchlichen Glaubenszustand;
- (Assertion OUT, Negation OUT) - die Position hat keinen Glaubenszustand.

In Abb. 2.12 verdeutlicht eine Darstellung einer TMS-Repräsentation und einer IBIS-Repräsentation die Unterschiede der beiden Formen. In Abb 2.13 ist die abstrakte Darstellung aus 2.12 in einem Beispiel instanziiert.

Das Beispiel verdeutlicht die oben ausgeführten Unterschiede zwischen einer IBIS-Darstellung und der issuebasierten Darstellung im TMS:

- Die verschieden Annahmen werden dem Issue *"Was machen wir?"* zugeordnet, das die Frage nach einer möglichen Freizeitbeschäftigung stellt.

- Die Annahme *"wir gehen schwimmen"* und *"wir gehen nicht schwimmen"* wird in der issuebasierten Form in einer einzigen Annahme zusammengefaßt. Die Begründung *"es ist nicht heiß"*, die die negierte Form der Annahme *"wir gehen nicht schwimmen"* unterstützt, wird als Argument dargestellt, daß die Position *"wir gehen schwimmen"* widerlegt.

2.3.9 Modellbasierte Repräsentation

Es handelt sich bei dieser Darstellungsform, im Gegensatz zu den bisher beschriebenen Systemen, nicht um die Darstellung von Argumentationsstrukturen. Die Beschreibung dieser Methode ist im Hinblick auf die Betrachtungen im Zusammenhang mit CoMo-Kit uninteressant. Sie erfolgt hier dennoch, um die Darstellung der verschiedenen DR-Methoden zu vervollständigen.

In einer modellbasierten Repräsentation wird Entwurfswissen in Form eines maschinenverständlichen Modelles gesammelt (daher die Bezeichnung **modellbasiert**). Diese Form der Darstellung eignet sich insbesondere für den Entwurf von Geräten. Ein Geräte Modell, als Softwaretool im Entwurfssystem integriert, liefert textuelle und graphische Erklärungen zum Produkt. Diese Information gibt Auskunft darüber, wie und warum das Produkt funktioniert.

Diese Form des DR bezeichnet Gruber [Gru90] als **DR durch Demonstration**: DR wird durch Erstellung einer Simulation gewonnen, die die Funktion der Struktur unter gegebenen Bedingungen erklärt.

Vorteil dieses Ansatzes ist es, daß die Dokumentation auf Anfrage generiert wird. Die vom Computer generierte Dokumentation wird aus dem formal dargestellten Wissen inferiert und erfolgt in natürlicher Sprache. Diese Art der Dokumentation ist interaktiv, da der Benutzer des Systems Fragen stellen kann und sich nicht auf die abgefaßten Erklärungen des Designers beschränken muß.

Ein ausführliches Beispiel zur Entwurfsdokumentation ist in Gruber 1990 [Gru90] dargestellt.

Die hier beschriebenen Ansätze zum Thema DR wurden völlig neutral beschrieben, auf eine Bewertung wurde verzichtet. Als nächstes soll nun der Einfluß dieser Formalismen auf die durchgeführten Erweiterungen von CoMo-Kit beschrieben werden. Erst der abschließende Vergleich in Kapitel 5 stellt die Ansätze aus diesem Kapitel zum erweiterten CoMo-Kit in Zusammenhang.

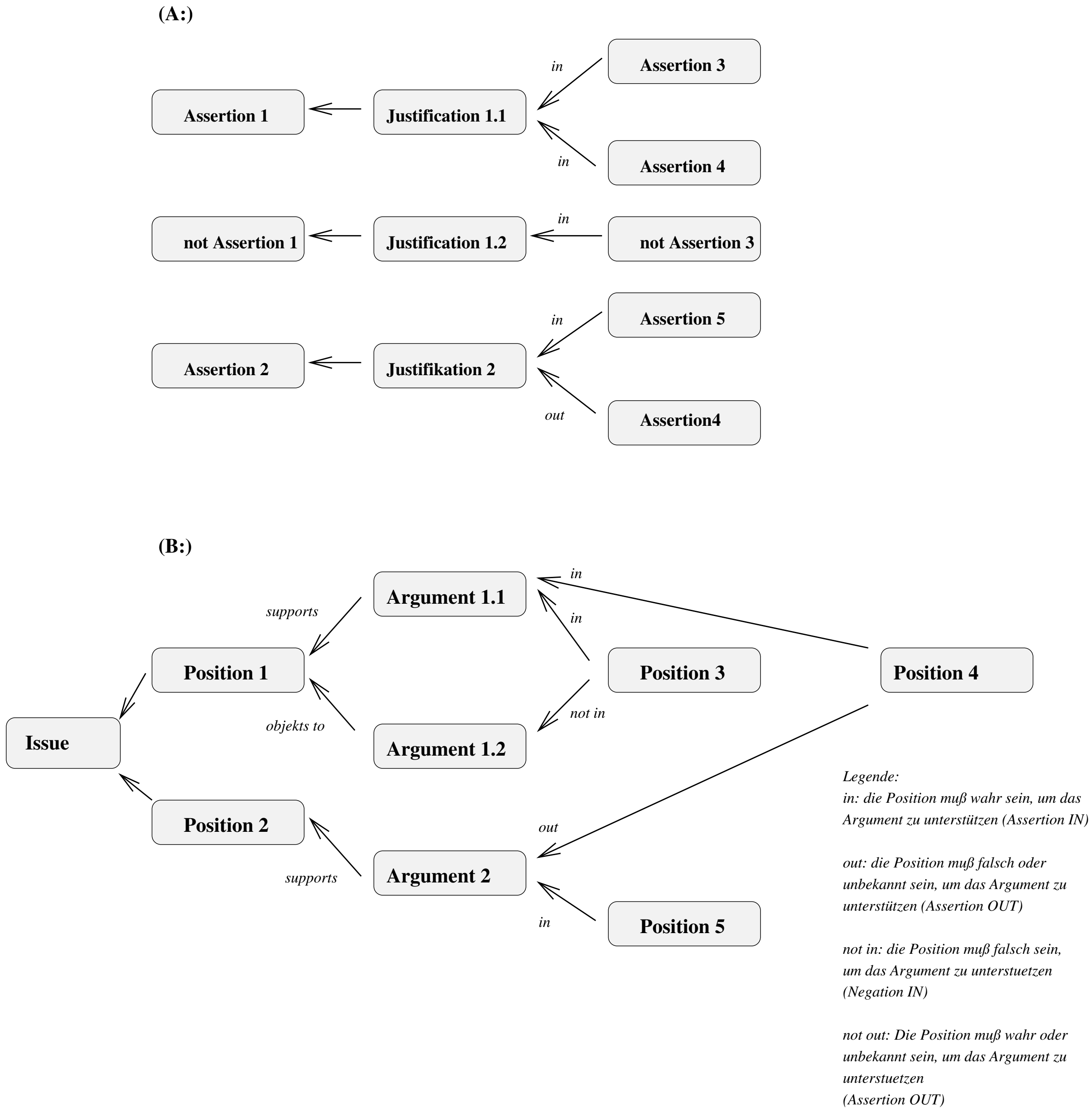

Abbildung 2.12: Vergleich zwischen **(A:)** einer TMS-Darstellung und **(B:)** einer erweiterter IBIS-Darstellung des gleichen Sachverhaltes. *Assertions* werden in der issuebasierten TMS-Darstellung durch *Positions* repräsentiert, *Justifications* durch *Arguments* (Quelle: [Lub91]).

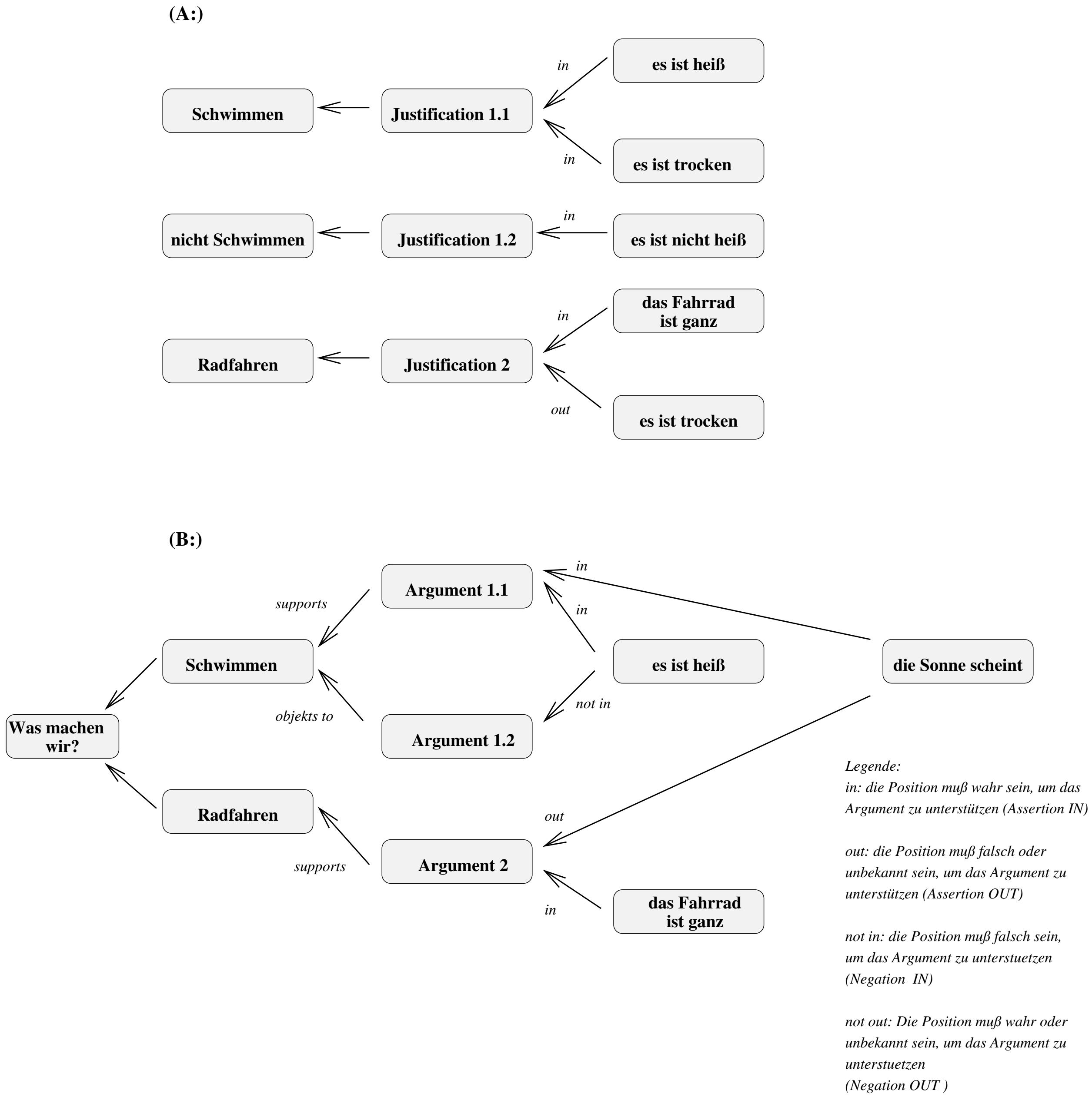

Abbildung 2.13: Vergleich zwischen **(A:)** einer TMS-Darstellung und **(B:)** einer erweiterter IBIS-Darstellung anhand eines Beispieles

Kapitel 3

Design Rationale in CoMo-Kit

In diesem Kapitel erfolgt zuerst eine allgemeine Einleitung zu CoMo-Kit, in der die wesentlichen Begriffe und Strukturen erläutert werden. Anschließend wird dargestellt, welche Erweiterungen von CoMo-Kit im Rahmen der Arbeit entwickelt und realisiert wurden. Der Abschnitt 3.3 stellt den Bezug zum Kapitel 2 her. Er erläutert, welche der in Kapitel 2 beschriebenen Elemente der Argumentation in CoMo-Kit identifiziert werden können, und welche Beziehungen zwischen ihnen bestehen.

3.1 Eine Einführung in CoMo-Kit

CoMo-Kit ist ein Werkzeug zur Modellierung, Steuerung und Abwicklung kooperativer Designprozesse. Mit Hilfe von Modellen werden die zu erledigenden Aufgaben, die Methoden zur Lösung der Aufgaben, sowie die entwickelten Produkte beschrieben. Die Modelle werden dann zur Durchführung des Entwicklungsprozesses operationalisiert. Dieses Kapitel wird das System CoMo-Kit nur recht oberflächlich beschrieben. Es enthält eine kurze Erläuterung der grundlegenden Begriffe, die zum weiteren Verständnis notwendig sind. Um tiefere Einblicke in CoMo-Kit zu erhalten, wird der interessierte Leser auf die entsprechende Literatur verwiesen [Mau93, Del94, DMP95].

In Abb. 3.1 ist die Grobarchitektur von CoMo-Kit dargestellt. Sie umfaßt die folgenden drei Komponenten:

Eine statische Wissensbasis in der die Modelle abgelegt sind (Grundlagen: [Mau93]; Erweiterungen: [Sch94]).

Modellierungswerkzeuge mit deren Hilfe die Modelle der Produkte und Prozesse erstellt oder verändert werden können (Grundlagen: [Mau93]; Erweiterungen: [Sch94, Sch95]).

Eine Ausführungskomponente(Interpreter) dient dazu, die im konzeptuellen Modell definierte Aufgabenstruktur in Interaktion mit dem Benutzer abzuarbeiten. Der Interpreter umfaßt eine Steuerungskomponente, den Scheduler, und die Clientprozesse. Das zu erstellende Produkt entsteht durch Interaktion der Clients mit dem Scheduler [Del94, Sch94].

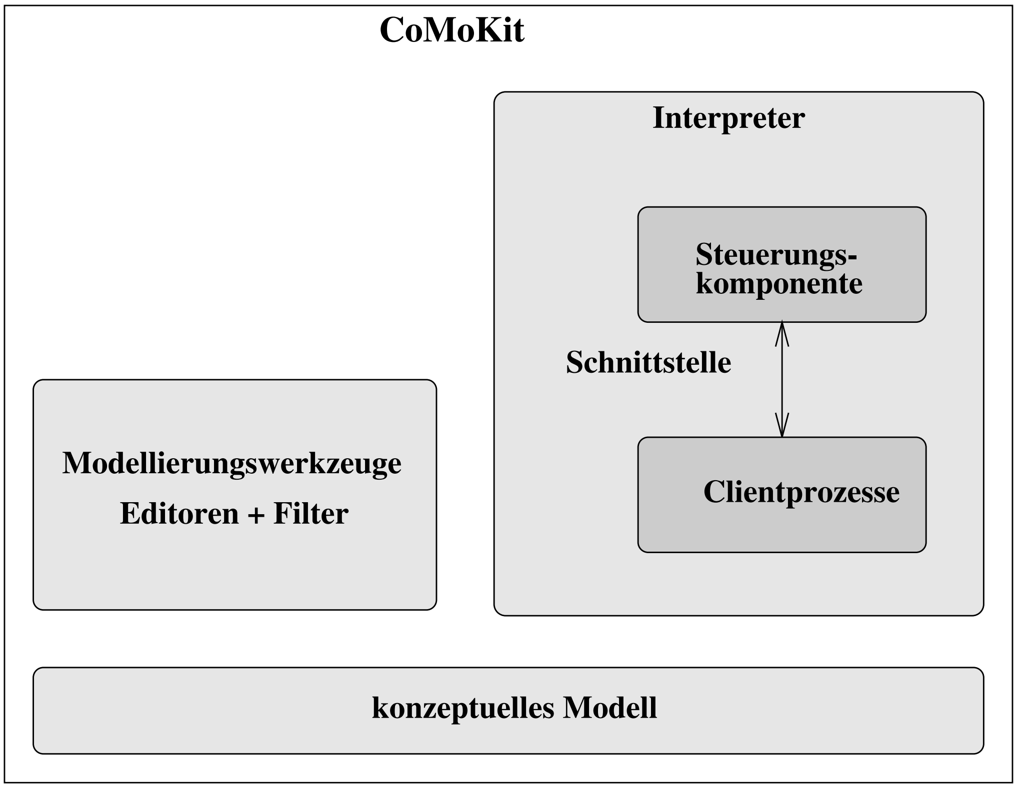

Abbildung 3.1: Die Grobarchitektur von CoMo-Kit (Quelle: [Del94])

Bei der Erstellung des Modells werden Elemente betrachtet, die im folgenden Abschnitt beschrieben werden. Wie die Modelle dann durch den Interpreter operationalisiert werden, ist in Abschnitt 3.1.2 dargestellt. Welche Abhängigkeiten diese Operationalisierung ausnutzt, wird im Abschnitt 3.3.2 erläutert.

3.1.1 Basisstrukturen

Mit den folgenden Elementen können die Modelle von kooperativen Designprozessen beschrieben werden:

Aufgaben (Tasks): Eine Task beschreibt ein zu erreichendes Ziel. Ihr sind Eingaben und Ausgaben zugeordnet. Bei der Bearbeitung einer Aufgabe wird eine Entscheidung getroffen, die sich in einer Aufgabenzerlegung oder der Zuweisung von Werten an die Ausgaben manifestiert. Unser Ansatz geht davon aus, daß zwischen den Eingaben und den Ausgaben ein kausaler Zusammenhang existiert. Dem liegt die Annahme zugrunde, daß bei der Erstellung des Prozeßmodells einer Aufgabe nur Eingaben zugeordnet werden, die für die Bearbeitung der Aufgabe notwendig sind (auf Unzulänglichkeiten dieser Annahme und deren Behebung wird in Kapitel 4 eingegangen).

Methoden: Um eine Aufgabe zu bearbeiten wird eine Methode angewandt. Für jede Aufgabe kann es eine Menge alternativer Methoden geben. Methoden werden von Agenten ausgeführt. Wir unterscheiden zwischen *atomaren* und *komplexen* (zusammengesetzten) Methoden. Atomare Methoden weisen Variablen aktuelle Werte zu. Eine komplexe Methode wird als Datenflußgraph beschrieben. Ein Datenflußgraph besteht aus Knoten, die (Unter-)Aufgaben und Variable definieren, und aus Kanten, die den Aufgaben ihre Ein- bzw. Ausgaben zuordnen. Jeder Variable besitzt einen Typ (entspricht einer Konzeptklasse). Dieser wird zur Laufzeit instanziiert und deren Instanz dann an die Variable gebunden wird. Jede Unteraufgabe kann mit Hilfe von Methoden weiter zerlegt werden. In diesem Sinn kann die Zerlegung der Gesamtaufgabe als UND-ODER-Baum verstanden werden. Bei diesem entsprechen Aufgaben den UND-Knoten, wohingegen Methoden ODER-Knoten sind. Die für die Lösung einer Aufgabe sinnvolle Methode wird im Rahmen der Projektplanung erst während der Projektdurchführung bestimmt.

Konzepte: Im Verlauf eines Entwurfsprozesses zu erzeugende Produkte werden mit einem objekt-zentrierten Ansatz modelliert, wobei zwischen Klassen und Instanzen unterschieden wird. Klassen definieren die Struktur der Instanzen durch eine Menge von Slots. Jedem Slot wird ein Typ und eine Kardinalität zugeordnet. Typen sind andere Konzeptklassen oder Basistypen wie SYMBOL, STRING, REAL, etc. Konzeptklassen werden benutzt, um die Variablen der Datenflußgraphen zu typisieren. In diesem Zusammenhang wird auch der Begriff *formaler Parameter* verwendet. Formale Parameter verknüpfen Konzeptklassen mit Aufgaben. Ein Parameter ordnet allen Aufgaben die Konzeptklasse zu, die dieselbe Instanz dieser Klasse als Ein- oder Ausgabe verwenden. Man kann den formalen Parameter mit einem Übergabeparameter in prozeduralen Programmiersprachen vergleichen.

Agenten: Aufgaben werden von Agenten bearbeitet. Im konzeptuellen Modell wird für jede Aufgabe definiert, welche Agenten fähig sind, sie zu bearbeiten. Wir unterscheiden dabei zwischen zwei verschiedenen Arten von Agenten: Menschen und Rechner. In Abhängigkeit von den zugeordneten Agenten werden Methoden in einer natürlichen oder formalen (operationalen) Sprache beschrieben.

3.1.2 Die Ausführungskomponente

Die Ausführungskomponente ist als Client-Server-Architektur realisiert. Abbildung 3.1.2 beschreibt deren Elemente im Überblick.

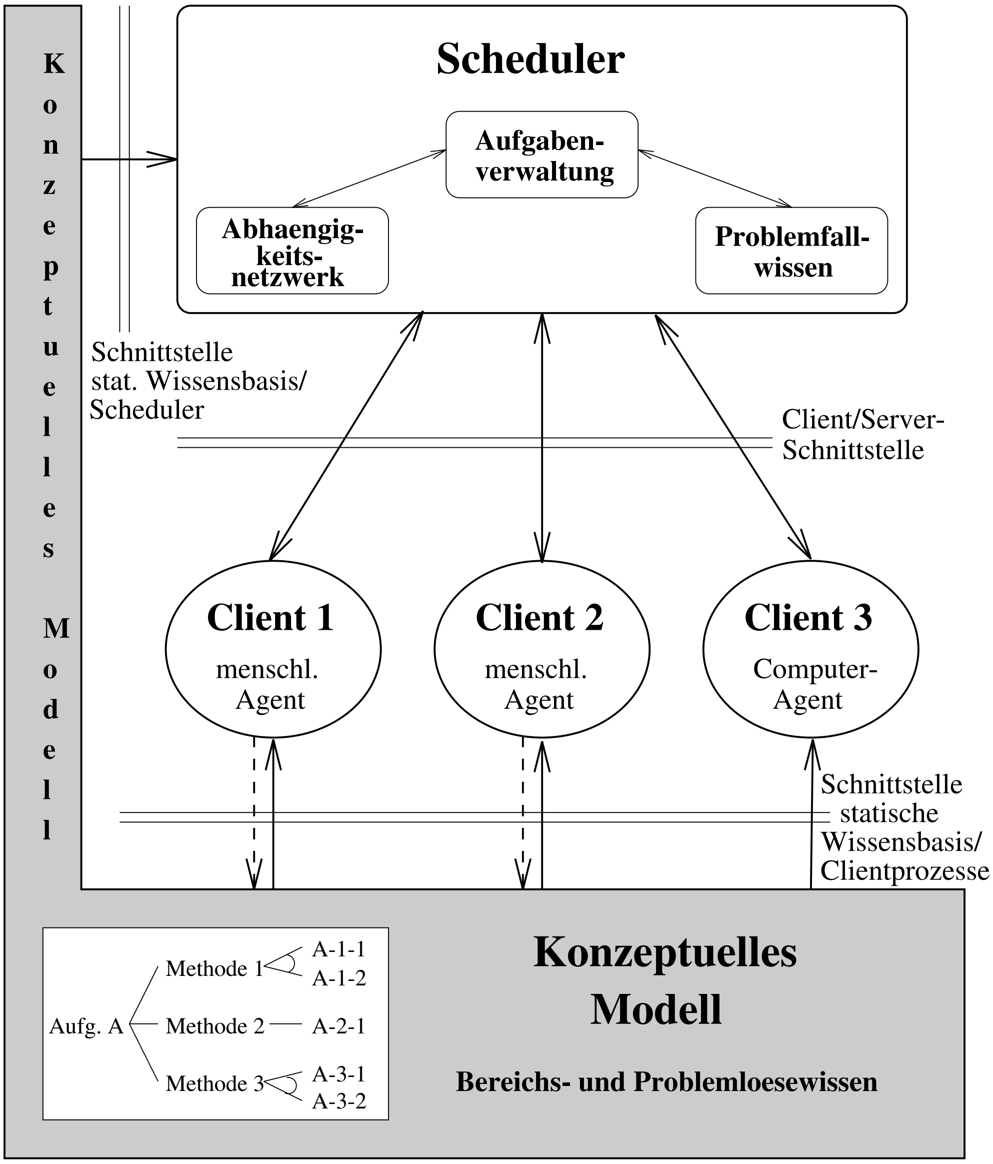

Abbildung 3.2: Die Grobarchitektur der Ausführungskomponente von CoMo-Kit (Quelle: [Del94])

Der Scheduler stellt den Server dar. Er steuert und verwaltet den Entwurfsprozeß, indem er die Agenten bei der Bearbeitung der Teilaufgaben anleitet. Der Scheduler übernimmt die folgenden Aufgaben:

- Er verwaltet die Abhängigkeiten zwischen Methoden und deren Ein- und Ausgaben. Diese werden als Datenflußabhängigkeit bezeichnet. Eine genauere Beschreibung dieser Abhängigkeiten erfolgt in Abschnitt 3.3.2.
- Er verwaltet die Abhängigkeiten, die sich durch die Aufgabenzerlegung ergeben.
- Er kontrolliert den Bearbeitungszustand der einzelnen Aufgaben und steuert deren Zustandsübergänge, die sich im wesentlichen durch die Aufgabenzerlegung, die Datenflußabhängigkeiten und die Wahl der Methoden zur Bearbeitung einer Aufgabe ergeben.
- Er überwacht die Konsistenz der Lösung, indem er die oben beschrieben Abhängigkeiten ausnutzt.

Die Clients führen den kooperativen Entwurfsprozeß aus, indem sie die einzelnen Aufgaben wie folgt bearbeiten:

- Sie wählen Methoden aus, die zur Bearbeitung eines Tasks angewendet werden können.
- Sie belegen die Ausgabevariablen atomarer Aufgaben, indem sie die entsprechende Aufgabe abarbeiten.
- Neue Aufgaben, die sich aus komplexen Methoden ergeben, werden zu den einzelnen Clienten delegiert, die am Entwurf beteiligt sind.

3.2 Motivation zur Verwendung von Design Rationale in CoMo-Kit

Die folgende Anforderung an CoMo-Kit gab den Anlaß, sich im Rahmen dieser Arbeit mit der Thematik Design Rationale zu beschäftigen:

Computerunterstützte Propagierung von Änderungen im Entwurf: Im Verlauf von komplexen Entwicklungsprozessen werden von den beteiligten Personen eine große Zahl von Einzelentscheidungen getroffen, die sich gegenseitig beeinflussen und aufeinander aufbauen. Eine Darstellung von Entscheidungen und den zugehörigen Begründungen macht es möglich, diese Abhängigkeiten explizit zu machen. Die Abhängigkeiten zwischen den einzelnen Entscheidungen erlaubt es Änderungen im Entwurf auf alle Entscheidungen fortzupflanzen, die von dieser Änderung betroffen sind. Der Anwender wird damit von der Aufgabe entlastet, zu analysieren, welche Teile des Entwurfes bei einer Überarbeitung betrachtet werden müssen.

Verständnis des bestehenden Entwurfes im Zusammenhang mit

- **der Überarbeitung des konzeptuellen Modelles:** Wenn Begründungen von Entscheidungen auch bei der Instanziierung des Modells beschrieben werden können, ist es möglich das konzeptuelle Modell bei einer Überarbeitung um diese Strukturen zu ergänzen. Das Modell wird dann nach mehreren Instanziierungen immer besser mit dem tatsächlichen Entwurfsprozeß übereinstimmen.
- **kooperativen und zeitlich ausgedehnten Projekten:** Sind mehrere Personen am Entwurfsprozeß beteiligt, so sollen die einzelnen Mitarbeiter in der Lage sein, die Entwurfsentscheidungen anderer Personen nachvollziehen zu können. Überarbeitet eine Person den eigenen Entwurfsprozeß, so sollten die Begründungen für oder gegen bereits getroffene Entscheidungen verfügbar sein, um sie nach längerer Zeit ins Gedächtnis zurückzurufen.

Diese Ziele zeigen Übereinstimmungen, zu den in Abschnitt 2.2 dargestellten Motivationen anderer Autoren. Wir finden sie in den dort aufgeführten Punkten 3 (Verständnis des Entwurfes) und 4 (Wirkung von Änderungen in Anforderungen oder Entwurf).

Die für CoMo-Kit beschriebenen Anforderungen machen deutlich, daß die Begründung von Entscheidungen im Mittelpunkt der Überlegungen stehen. Unsere Zielsetzung geht dabei über die Funktionalität der im Kapitel 2 beschriebenen Systeme hinaus. Unser Ziel ist nicht nur eine Darstellung des Argumentationsraumes, sondern insbesondere die automatische Propagierung von Änderungen im Entwurf. Ist die automatische Propagierung von Änderungen möglich, so müssen die zu diesem Zweck verwendeten Abhängigkeiten nur noch entsprechend aufgearbeitet werden, um deren Darstellung für den Anwender zu ermöglichen. Umgekehrt ist eine Verwaltung der Abhängigkeiten erst möglich, wenn die entsprechenden Elemente der Argumentation auch repräsentiert werden. Diese wechselseitige Beziehung verdeutlicht den Zusammenhang der beiden gewünschten Anforderungen.

Da diese Funktionalität als Erweiterung in ein bereits bestehendes Netz von Abhängigkeiten integriert werden soll, werden in den nächsten Abschnitten die folgenden Schritte beschrieben:

1. In Abschnitt 3.3 werden die Elemente der Argumentation (Entscheidungen, Begründungen, ...) in CoMo-Kit identifiziert und zu den Strukturen aus der IBIS-Notation in Zusammenhang gebracht. Der Vergleich der Strukturen bezieht sich auf die IBIS-Notation, weil diese Elemente in den meisten Systemen aufgegriffen werden. Sie beschreiben die grundlegenden Elemente einer argumentativen Darstellung.

   Außerdem wird im Abschnitt 3.3 beschrieben, welche Abhängigkeiten zwischen diesen Elementen im bisherigen CoMo-Kit bestehen.

Der Abschnitt gibt also keine Ergebnisse dieser Arbeit wieder, sondern betrachtet die Arbeiten von Maurer, Dellen und Schmitz unter einem argumentationsbasierten Blickwinkel.

2. Auf dieser Grundlage wird dann in Kapitel 4 erläutert, welche Elemente zur Darstellung einer Argumentation noch ergänzt werden müssen, und wie die bestehenden Abhängigkeiten variiert werden müssen, um die gewünschte Funktionalität zu erhalten.

3.3 Darstellung von Entscheidungen, Begründungen und Abhängigkeiten in CoMo-Kit

3.3.1 Die Elemente der Argumentation in CoMo-Kit

Bevor die Elemente der Argumentation in CoMo-Kit identifiziert werden, muß deutlich sein, was der Begriff *Begründung* in diesem Zusammenhang bedeutet.

In unserem Verständnis beschreibt eine Begründung die Rechtfertigung für die Akzeptanz oder Ablehnung einer Aussage. Welche Bedeutung die Begründung für die Gültigkeit der Aussage hat, hängt von der Auswertungsstrategie ab, mit der Begründungen bewertet werden, um den Zustand der Entscheidung zu bestimmen. In CoMo-Kit müssen alle Begründungen, die die Aussage unterstützen, erfüllt sein, damit die Aussage gültig ist. Gleichzeitig darf keine Begründung für das Ablehnen der Aussage zutreffen. Der Zusammenhang der Begründungen und Aussagen kann also mit Hilfe der logischen Folgerung dargestellt werden.

Nun sollen die Elemente Entscheidung und Begründung in CoMo-Kit identifiziert werden, um festzustellen, inwiefern die Grundlagen einer argumentativen Darstellung in CoMo-Kit bereits vorhanden sind:

- Der Entwickler muß eine bestimmte *Aufgabe* lösen, zu dessen Bearbeitung ihm meist verschiede *Methoden* zur Auswahl stehen. Eine Aufgabe entspricht einem *Issue* im argumentativen Sinne der IBIS-Notation (vgl. Abschnitt 2.3.2). Eine *Task* beschreibt, ähnlich einem Issue, ein zu erreichendes Ziel.

- Die Methoden, die zur Lösung der Aufgabe zur Verfügung stehen, entsprechen den *Positions* von IBIS. Der Anwender wählt eine Methode aus einer Menge von Alternativen. Die Wahl einer Methode beschreibt eine Entscheidung. Die Methode, die im aktuellen Lösungskontext gültig ist, repräsentiert die *gewählte Position* in IBIS.

  – Bei der Wahl einer komplexen Methode, entscheidet sich der Entwickler für eine bestimmte Vorgehensweise.

  – Die Wahl einer atomaren Methode repräsentiert die Erzeugung eines bestimmten Produktes.

- Die Gründe für die Wahl einer Methode, sind Begründungen der Entscheidungen. Sie entsprechen den *Arguments* in IBIS. Die einzigen Begründungen, die im bisherigen CoMo-Kit-System vorgegeben sind, sind die Eingabevariablen für atomare Methoden. Sie stellen Argumente für die Wahl einer Methode dar. Sie werden durch den Datenfluß vorgegeben und sind durch eine vorangegangene Methodenanwendung entstanden, sind also Produkte des Entwurfsprozesses.

Dieser Vergleich zwischen den CoMo-Kit-Elementen und den IBIS-Strukturen verdeutlicht, daß CoMo-Kit, im Unterschied zu IBIS, nicht beliebige Elemente als Issues, Positions und Arguments besitzt.

Ein Issues, und damit der eigentliche Gegenstand der Diskussion, ist immer ein Aufgabe des Entwurfsprozesses. Verschiedene Positions zur Lösung des Issues sind immer die möglichen Methoden zur Lösung dieser Aufgaben. In CoMo-Kit werden nur Entscheidungen innerhalb von Entwurfsprozessen beschrieben, während IBIS beliebige Entscheidungen beschreiben kann.

Als Argumente wurden bisher nur Produkte des Entwurfsprozesses verwendet. Wir werden in Kapitel 4 sehen, daß es sinnvoll ist, diese Einschränkung von CoMo-Kit im Vergleich zu IBIS zu beheben.

Tasks, Methods und formale Parameter sind Strukturen des konzeptuellen Modells. Bei der Instanziierung des Modells werden diese Elemente in ein Netz übertragen, das die Abhängigkeiten des Projektes verwaltet. In diesem Netz werden die Elemente wie folgt repräsentiert:

Entscheidungen: Eine Entscheidung für eine Methode m_i wird durch das Prädikat $decision(m_i)$ beschrieben. Hat das Prädikat den Wert *true*, so ist die Entscheidung gültig, hat es den Wert *false*, so ist die Entscheidung nicht gültig. Die Notwendigkeit des Rückzuges von Entscheidungen wird durch das Prädikat $rejected\_decision(m_i)$ ausgedrückt. Die Gültigkeit des Prädikates bedeutet, daß die Methode m_i für den aktuellen Lösungsprozeß nicht angewendet werden kann.

Begründungen durch Eingabevariablen werden durch das Prädikat $assignment(Xi = x_i)$ beschrieben. Die Gültigkeit diese Prädikates bedeutet, daß ein Produkt, welches durch die Variable X_i repräsentiert wird, an den aktuellen Wert x_i gebunden ist.

Im folgenden Abschnitt werden nun die Abhängigkeiten zwischen diesen Objekten beschrieben.

3.3.2 Abhängigkeiten in CoMo-Kit

Das konzeptuelle Modell beschreibt Beziehungen zwischen Aufgaben, Methoden und Produkten, die bei der Abwicklung des Entwurfsprozesses berücksichtigt werden müssen. Sie werden bei der Operationalisierung des Modelles zum Aufbau eines

Abhängigkeitsnetzwerkes verwendet. Der Scheduler nutzt diese Information dann zur Steuerung des Prozesses.

Folgende Abhängigkeiten werden unterschieden:

Abhängigkeiten durch den Datenfluß: Bei der Bearbeitung einer Aufgabe werden Produkte konsumiert und produziert. Diese Beziehung wird im konzeptuellen Modell beschrieben und läßt sich in Abhängigkeiten für den Lösungsprozeß übertragen.

- **Abhängigkeit beim Erzeugen von Information:** Die Anwendung einer atomaren Methode zur Lösung der Aufgabe bestimmt, welche konkreten Werte an die Ausgabevariablen einer Task gebunden werden. Bei der Prozeßabwicklung wird nun das Ergebnis dieser atomaren Methodenanwendung an die Entscheidung für diese Methode geknüpft. Mit einem Rückzug der Entscheidung verliert die resultierende Variablenbelegung ihre Gültigkeit.

  Dieser Zusammenhang wird formal durch die folgende Formel beschrieben:

$$\begin{aligned} &decsion(m_i) \Rightarrow \\ &assignment(O_1 = o_{1k}) \wedge \ldots \wedge assignment(O_n = o_{nk}) \end{aligned} \tag{3.1}$$

- **Abhängigkeiten beim Konsumieren von Information:** Durch das konzeptuelle Modell wird beschrieben, welche Informationen zur Bearbeitung einer Task benötigt werden. Diese Informationen beeinflussen die aktuelle Entscheidung und werden deshalb auch als Begründungen der Entscheidung betrachtet.

  Auf logischer Ebenen wird dies wie folgt beschrieben:

$$\begin{aligned} &\neg assignment(I_i = i_{1k}) \vee \ldots \vee \neg assignment(I_n = i_{nk}) \\ &\Rightarrow rejected\_decision(m_{neu}) \end{aligned} \tag{3.2}$$

  Die Informationen selbst sind Produkte vorangegangener Methodenanwendungen, beruhen also auf vorangegangen Entscheidungen. Werden diese vorangegangen Entscheidungen ungültig, so verliert damit die Entscheidung die durch die Information begründet wurde, ihre Begründung und wird ebenfalls ungültig. Die Begründungen werden an das Prädikat *rejected_decision* gekettet, um damit die Notwendigkeit des Rückzuges auszudrücken. Wir werden auf den Zusammenhang zwischen einer Entscheidung und deren Rückzug noch etwas genauer eingehen.

Abhängigkeiten durch die Aufgabenzerlegung: Im konzeptuellen Modell werden Aufgaben durch komplexe Methoden in Teilaufgaben zerlegt und definieren dadurch Abhängigkeiten. Die Entscheidung für die komplexe Aufgabe impliziert die Gültigkeit der daraus resultierenden Teilaufgaben. Um zu beschreiben, daß eine Aufgabe zum aktuellen Plan gehört, wird das Prädikat $validSubTask(T_1)$

definiert. Das Prädikat ist gültig, wenn das Ergebnis in den Entwicklungsprozeß eingeht. Der beschriebene Zusammenhang läßt sich somit durch die folgende Formel dargestellt:

$$decision(m_i) \Rightarrow validSubTask(T_1) \wedge \ldots \wedge validSubTask(T_n) \tag{3.3}$$

Wird diese komplexe Methode aus irgendeinem Grund zurückgezogen, so dürfen auch die daraus resultierenden Teilaufgaben nicht mehr gültig sein. Entscheidungen, die auf dieser Zerlegung aufbauen, müssen ebenfalls zurückgenommen werden. Dieses Verhalten wird durch eine Propagierung erreicht. Die Entscheidung $decision(m_i)$ wird ungültig, wenn die zugehörige Task ungültig wird (wenn das Prädikat $validSubTask$ ungültig wird).

Abhängigkeit zwischen Gültigkeit und Rückzug einer Entscheidung:
Wird eine Entscheidung zurückgezogen, so ist das Prädikat $rejected\_decision(m_i)$ gültig. Das Prädikat $decision(m_i)$ wird dann ungültig, weil die Entscheidung nicht mehr zum aktuellen Lösungsweg gehört. Dieser Zusammenhang wird durch die folgende Formel beschrieben:

$$rejected\_decision(m_i) \Rightarrow \neg decision(m_i) \tag{3.4}$$

Es ist wichtig zu betonen, daß zwischen dem Prädikat $rejected\_decision$ und $\neg decision$ keine Äquivalenzrelation besteht. Das Prädikat $rejected\_decision$ repräsentiert die Notwendigkeit eines Rückzuges, nicht den eigentlichen Rückzug. Diese Unterscheidung ist sinnvoll, um das folgende Verhalten des Systems zu erreichen.

Wurde eine Entscheidung zurückgezogen und stellt sich deren Rückzug zu einem späteren Zeitpunkt als unbegründet heraus (die Notwendigkeit des Rückzuges ist nicht mehr gegeben), so soll die aktuelle Lösung beibehalten werden. Der Rückzug soll nicht rückgängig gemacht werden, weil der aktuelle Lösungsweg unter Umständen schon recht weit fortgeschritten ist. Würde der oben beschriebene Zusammenhang durch eine Äquivalenzrelation beschrieben, so würde die Entscheidung wieder gültig werden, sobald die Notwendigkeit des Rückzuges hinfällig wäre.

Das aus in Gleichungen 3.1 bis 3.4 dargestellten Abhängigkeiten werden durch ein TMS verwaltet. Sie werden bei der Operationalisierung des Prozeßmodelles automatisch in ein Abhängigkeitsnetzwerk übertragen, das die beschriebene Verkettung von Produkten und Entscheidungen abbildet.

Dieses TMS beruht auf einer erweiterten Implementierung von REDUX [Rit93] (vgl. 2.3.7). Die Darstellung der Abhängigkeiten mittels REDUX ermöglicht es, die Änderungen im Entwurf zu propagieren.

Die beschriebenen Prädikate werden im TMS durch Knoten dargestellt, die entsprechend der Gültigkeit der Prädikate IN oder OUT markiert sind. Der Bedeutung der Formeln 3.1 und 3.4 wird durch eine entsprechende Verwaltung der Knoten in IN- und OUT-Listen realisiert. Um eine detaillierte Beschreibung der Zusammenhänge zu erhalten, wird der interessierte Leser auf die Arbeit von Dellen [Del94] verwiesen.

Vergleich der Abhängigkeiten mit der IBIS-Darstellung

Im folgenden wird nun dargestellt in wie weit die generierten Abhängigkeiten mit den IBIS-Beziehungen übereinstimmen.

- **Beziehungen zwischen Aufgaben und Teilaufgaben** entsprechen den IBIS-Beziehungen zwischen *Issues*.
- **Beziehungen zwischen Aufgaben und Methoden** entsprechen den *is-suggested-by*-Relationen zwischen *Issues* und *Positions* in IBIS.
- **Beziehungen zwischen atomaren Aufgaben und Eingabevariablen** entsprechen der *supports*-Relationen zwischen *Arguments* und *Positions* in IBIS.

Diese Ausführungen machen deutlich, daß in CoMo-Kit wesentlich weniger Beziehungen beschrieben werden als in IBIS. So fehlt unter anderem die Beziehung *objects-to*, um darzustellen, daß ein Argument eine Entscheidung widerlegt. Im folgenden Kapitel werden wir darstellen, um welche Beziehungen CoMo-Kit aus diesem Grund erweitert wurde.

Die Abbildungen 3.3 und 3.4 stellen die Elemente und Beziehung von CoMo-Kit und IBIS noch einmal zusammenfassend dar. Abbildung 3.3 stellt die Beziehungen aus der Aufgabenzerlegung dar. Abbildung 3.4 beschreibt die Abhängigkeiten aus dem Datenfluß. Die Darstellung verdeutlicht, daß CoMo-Kit auch Abhängigkeiten beschreibt, die durch eine IBIS-Darstellung nicht erfaßt werden. CoMo-Kit geht beschränkt sich nicht auf die argumentative Darstellung sonder beschreibt ebenfalls Information aus Produkt und Prozeß. Kapitel 5 behandelt dieser Unterschied zwischen CoMo-Kit und den anderen Systemen etwas genauer.

Nachdem nun die Elemente und Funktionen von CoMo-Kit erläutert wurden, beschreibt das folgende Kapitel, welche Modifikation an CoMo-Kit vorgenommen wurde.

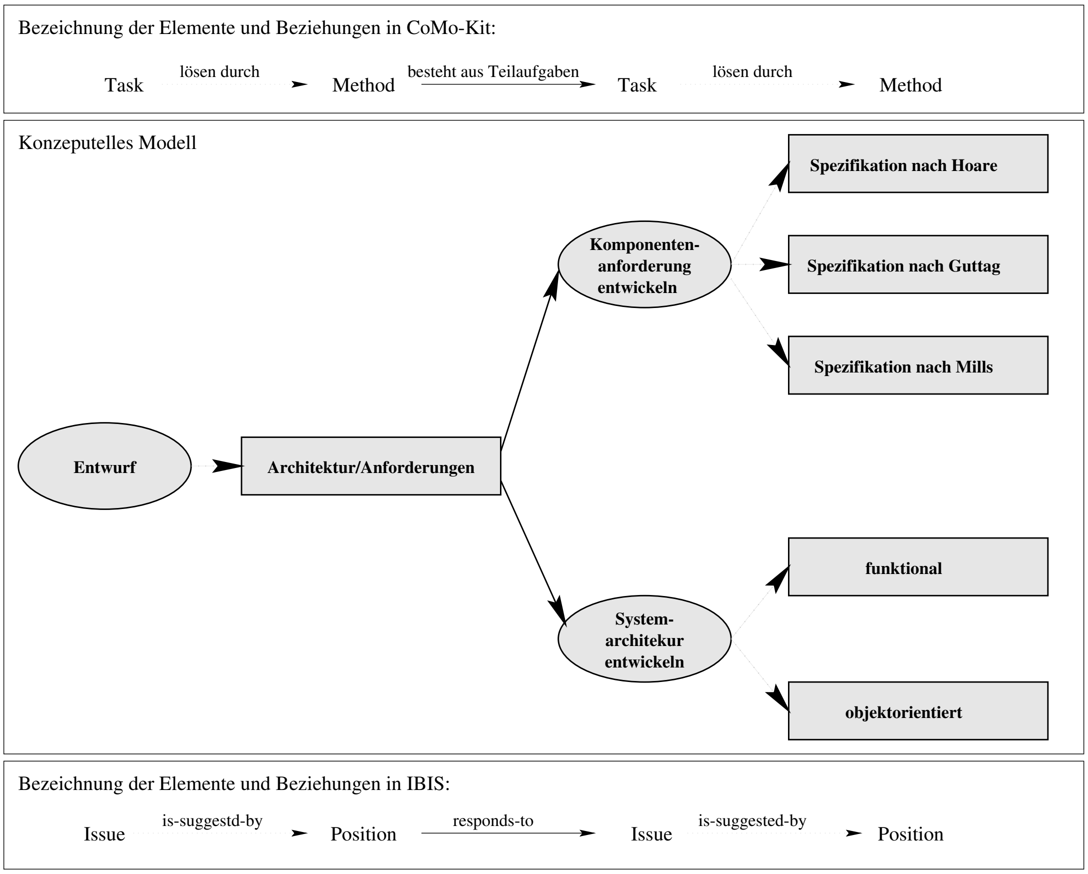

Abbildung 3.3: Zusammenhang zwischen IBIS und CoMo-Kit-Elementen: Darstellung der Aufgabenzerlegung

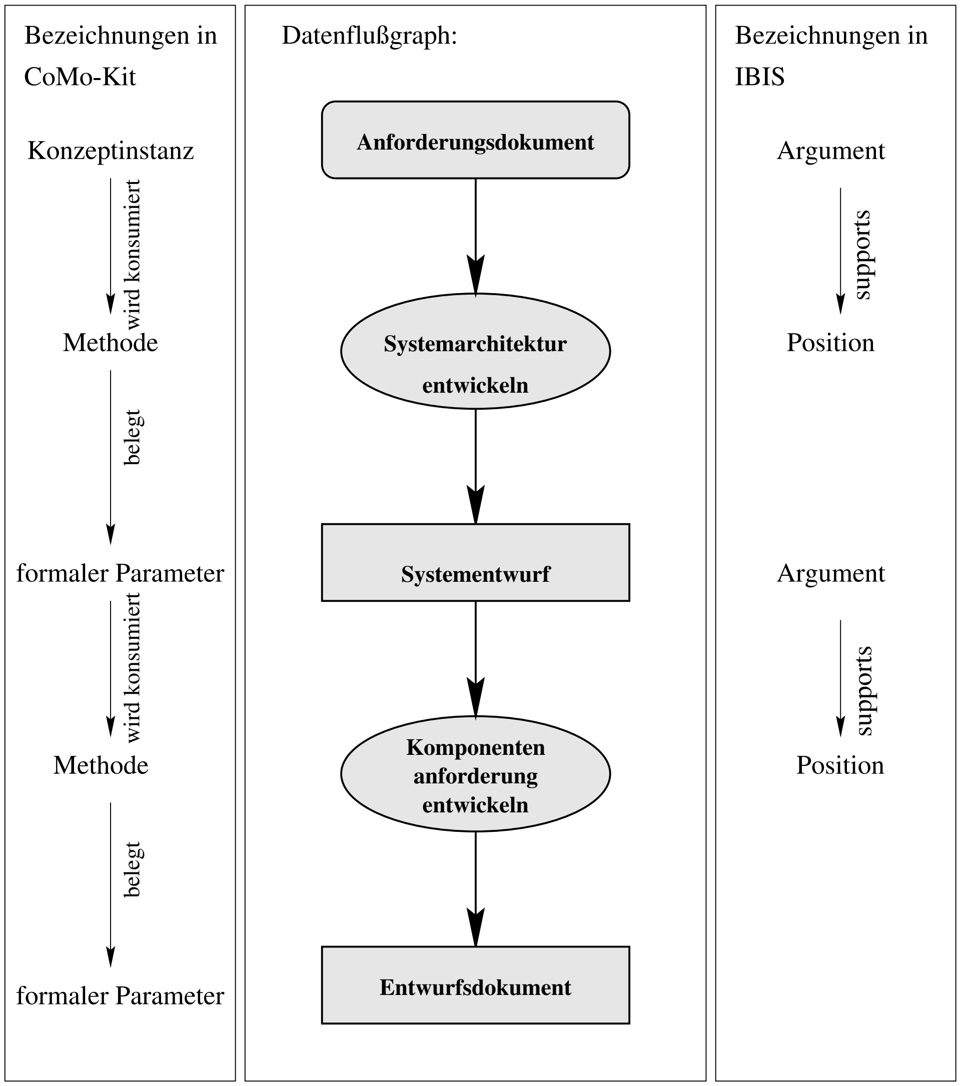

Abbildung 3.4: Zusammenhang zwischen IBIS und CoMo-Kit-Elementen: Darstellung des Datenflusses

Kapitel 4

Die Erweiterung von CoMo-Kit

Im folgenden Abschnitt wird dargelegt, inwieweit die beschriebenen Abhängigkeiten und Begründungen ausreichen und wie sie modifiziert werden müssen.

Die bisher enthaltenen Begründungen in CoMo-Kit und die beschriebenen Abhängigkeiten werden durch CoMo-Kit automatisch aus dem Prozeßmodell generiert. Damit ist der Anwender von einem großen Teil der Dokumentation befreit. In der Praxis zeigt es sich, daß es erforderlich ist, zusätzliche Begründungen eingeben zu können, und daß Prozeßmodelle unter Umständen nicht alle oder überflüssige Abhängigkeiten vorgeben.

Folgende Erfahrungen aus den bisherigen Domänen von CoMo-Kit haben dies verdeutlicht:

1. Information, die der Entwickler aus Erfahrung und Hintergrundwissen bezieht, die aber nicht in Form von Eingabevariablen durch den Datenfluß beschreiben ist, kann bisher nicht in Form einer Begründung angegeben werden.

2. Entscheidungen können als Begründungen Produkte des Entwurfsprozesses heranziehen, die keine Eingabevariablen der entsprechenden Methode darstellen. Der Anwender kann diese Begründungen nicht angeben.

3. Durch das konzeptuelle Modell werden nur Gründe für die Wahl einer bestimmten Methode vorgegeben. Der Anwender kann keine Gründe für den Rückzug einer bestimmten Entscheidung angeben.

4. Häufig werden Entscheidungen durch vorangegangene Entscheidungen begründet. Der Anwender kann diesen Zusammenhang in CoMo-Kit nicht beschreiben.

5. Manchmal sind nur ein Teil, der durch das Modell vorgegebenen Eingangsvariablen der Methoden, wirklich für die Entscheidungen von Bedeutung. Das konzeptuelle Modell soll aber nicht prinzipiell auf nur diese Eingabevariablen eingeschränkt werden, weil sonst für jedes Projekt ein eigenes Modell notwendig wäre.

6. Erst durch die Belegung einer Ausgabevariablen, mit einem bestimmten Wert, wird erkennbar, daß diese Ausgabevariable als Begründung herangezogen wird.

Diese Unzulänglichkeit der Modelle lässen sich durch die folgenden Fälle zusammenfassen [1]:

- Modelle stellen generische Strukturen dar, die auf die konkrete Projektsituation nicht optimal zugeschnitten sind. Manche Modelle geben zu wenige Abhängigkeiten an, wir sprechen dann von **unvollständigen Modellen** (vgl. Punkte 1,2,3,4) . Andere Modelle beschreiben zu viele Abhängigkeiten, solche Modelle bezeichnen wir als **überladene Modelle** (vgl. Punkt 5).
- Ein detailliertes Prozeßmodell ist an vielen Stellen von den Ergebnissen vorangegangener Schritte der Projektdurchführung abhängig. Wurde zum Beispiel bei der Entwicklung einer bestimmten Komponente eine vorgegebene Anforderung aus dem Anforderungsdokument berücksichtigt, so ist diese Abhängigkeit bei der Erstellung des Prozeßmodelles noch nicht bekannt, sondern wird erst bei der Prozeßdurchführung deutlich. Wir bezeichnen dieses Phänomen als **zustandsabhängige Entscheidungen**(vgl. Punkt: 6).

Als Folge dieser Unzulänglichkeiten, muß das System dem Anwender die Möglichkeit zur Verfügung stellen, die bestehenden Abhängigkeiten während der Durchführung des Entwicklungsprozesses dynamisch zu manipulieren. Von diesen Veränderungen sind nur die Abhängigkeiten durch den Informationsfluß betroffen. Wir konzentrieren uns in diesem Zusammenhang auf die Begründung von Entwurfsentscheidungen, das heißt auf Begründungen für oder gegen die Wahl einer bestimmten Methode.

Die geforderten Anpassungen müssen in zwei Richtungen möglich sein:

- Unvollständige Modelle und zustandsabhängige Entscheidungen verlangen eine Erweiterung bestehender Abhängigkeiten um zusätzlichen Begründungen.
- Bei überladenen Modelle müssen die Abhängigkeiten aus dem Informationsfluß, die durch das Modell vorgegen sind (vgl. Abschnitt 3.3.2), zurückgenommen werden. Dies ist notwendig, um die Propagierung von Änderungen auf nur die Teilprozesse zu beschränken, die tatsächlich von der Änderung betroffen sind.

In den folgenden Abschnitten zeigen wir auf, wie die dynamische Anpassung während der Prozeßabwicklung durchgeführt wird. In Abschnitt 4.1 wird beschrieben, wie die Erweiterung bestehender Begründungen realisiert wird. Zu diesem Zweck stellen wir dar, welche unterschiedlichen Begründungen wir unterscheiden und welche Repräsentationsform für eine Begründung gewählt werden kann. Im Abschnitt 4.2 werden die Anpassungen zur Löschung bestehender Abhängigkeiten ausgeführt.

[1] Die beschriebene Begriffsbestimmung wurde in Rahmen dieser Arbeit entwickelt

4.1 Erweiterung bestehender Begründungen

4.1.1 Darstellung zusätzlicher Begründungen

Wir unterscheiden zwei Arten von Erweiterungen bestehender Begründungen:

- Gründe für die Wahl einer bestimmten Vorgehensweise.
- Gründe für das Verwerfen einer bestimmten Vorgehensweise.

Sprechen wir im folgenden allgemein von Begründungen, so verstehen wir darunter die beiden oben beschriebenen Typen. Wollen wir die beiden Begründungen explizit unterscheiden, so bezeichnen wir sie als *Begründungen der Gültigkeit* und *Begründungen des Verwerfens*

Zur Beschreibung zusätzlicher Begründungen stehen dem Anwender verschiedene Möglichkeiten zu Verfügung:

- Begründung durch vorangegangene Entscheidungen.
- Begründungen durch bestehende Produkte.
- Informelle textuelle Begründungen.

Im folgenden werden die einzelnen Formen erläutert. In diesem Zusammenhang soll auch deutlich werden, aus welchem Grund die Unterteilung in verschieden Formen der Darstellung sinnvoll ist und welche Restriktionen für die einzelnen Begründungen bestehen.

Begründungen durch vorangegangene Entscheidungen: Entscheidungen können durch bereits getroffene Entscheidungen beeinflußt werden. Zum Beispiel kann sich ein Agent für einen *"funktionalen Entwurf"* entschließen, weil in einer früheren Phase der Projektdurchführung entschieden wurde, die Methode *"funktionale Analyse"* zu wählen. In diesem Fall ist die Entscheidung für die Methode *"funktionale Analyse"* die Begründung der Entscheidung *"funktionaler Entwurf"*.

Eine Begründung durch Entscheidungen macht aber nur Sinn für Entscheidungen, die bereits getroffen wurden und außerdem gültig sind. Zusätzlich müssen diese Entscheidungen von der zu begründenden Entscheidung unabhängig sein. Das bedeutet, es dürfen keine Entscheidungen ausgewählt werden, die erst aus der zu begründenden Methode durch Aufgabenzerlegung hervorgehen. Ebensowenig darf die Begründung im Datenfluß von der zu begründenden Entscheidung abhängig sein. Sie darf also auch keine Produkte zu ihrer Bearbeitung benötigen, die aus der zu begründenden Methode generiert werden. Eine entsprechende Benutzeroberfläche stellt sicher, daß nur zulässige Entscheidungen auch als Begründungen ausgewählt werden dürfen.

Begründungen durch bestehende Produkte: Entscheidungen können durch bereits existierende Variablenzuweisungen beeinflußt werden. Diese wurden in einer vorangegangenen Phase des Prozesses als Ergebnis einer Methodenanwendung erzeugt. So kann sich zum Beispiel ein Agent für ein Vorgehen nach der Methode *"objektorientierter Entwurf"* entscheiden, weil im Anforderungsdokument als *"Anforderung-No.5: Implementierung in Smalltalk"* gefordert wird. In diesem Fall ist die Entscheidung für die Methode *"objektorientierter Entwurf"* durch ein Produkt, nämlich *"Anforderung-No.5"* begründet. In diesem Fall sind nur Produkte sinnvoll, die zum aktuellen Lösungsprozeß gehören.

Sowohl bei der Begründung durch Entscheidungen, als auch bei der Begründung durch bestehende Produkte, existieren die Begründungen als formale Elemente in unserem System. Die Entscheidungen werden durch die Decision-Nodes die Produkte durch die Assignment-Nodes dargestellt. Der Anwender kann bei der Begründung direkt auf diese Elemente verweisen, die ihm durch eine entsprechende Oberfläche präsentiert werden. Durch die Referenzierung auf bestehende Elemente, wird die Grundlage für eine computerunterstützte Reaktion auf Änderungen im Entwurf geschaffen. Ändert sich in unserem obigen Beispiel die *"Anforderung-No.5"*, so ist unser System in der Lage, diese Information auf die Entscheidung *"objektorientierter Entwurf"* mit Hilfe der in Abschnitt 3.3.2 beschriebene Technik weiter zu propagieren.

Informelle textuelle Beschreibung: Der Benutzer kann als Begründung einen Text eingeben. Diese Form ist für die Arten von Begründungen vorgesehen, die nicht durch vorangegangenen Entscheidungen oder Produkte beschrieben werden können. Diese Begründungen von Entwurfsentscheidungen können schnell eingegeben und für den weiteren Entwurfsprozeß konserviert werden. Der Text wird in Form eines Hypertextes der entsprechenden Entscheidung zugeordnet. Die textuelle Information kann keine automatische Propagierung von Änderungen unterstützen. Ihre Bedeutung liegt allein in der Archivierung der Information für die Benutzer.

Im folgenden Abschnitt werden wir erläutern, welche Auswirkung die zusätzlichen Begründungen auf die in Abschnitt 3.3 beschriebenen Zusammenhänge haben. Wir werden dies durch eine Modifikation der dort aufgeführten logischen Formeln verdeutlichen. Dabei werden nur die Begründungen durch vorangegangene Entscheidungen oder bestehende Wertzuweisungen betrachtet. Denn nur diese haben eine Bedeutung bei der Propagierung von Änderungen. Die textuellen Begründungen bleiben bei diesen Betrachtungen außen vor.

Der Anwender wird mit der logischen Ebene nicht konfrontiert, sondern kann die Abhängigkeiten über entsprechende Schnittstellen manipulieren. Eine Darstellung und Beschreibung dieser Schnittstelle erfolgt im Kapitel 6 anhand eines Beispiels und im Anhang I dieser Arbeit.

4.1.2 Erweiterung um Begründungen für die Gültigkeit einer Entscheidung

Begründung durch vorangegangene Entscheidungen: Eine aktuelle Entscheidung $decision(m_{neu})$ wird auf Grund einer oder mehrerer zurückliegender Entscheidungen $decision(m_{alt1})$ ... $decision(m_{altM})$ getroffen. Die Entscheidung $decision(m_{neu})$ soll ungültig werden, wenn mindestens eine der Entscheidungen $decision(m_{alt1}) \ldots decision(m_{altM})$ ihre Gültigkeit verliert. Um diesen logischen Zusammenhang formal zu beschreiben, wird Gleichung 3.2 um die Disjunktion der Negation der Prädikate $decision(m_{alt1}) \ldots decision(m_{altM})$ erweitert.

Die zusätzlichen Begründungen für die Gültigkeit einer Entscheidung verändern die Bedingungen für den Rückzug aus Gleichung 3.2 wie folgt: [2]

$$\begin{aligned} &\neg assignment(I_i = i_{1k}) \vee \ldots \vee \neg assignment(I_n = i_{nk}) \vee \\ &\neg \mathbf{decision(m_{alt1})} \vee \ldots \vee \neg \mathbf{decision(m_{altM})} \\ &\Rightarrow rejected - decision(m_{neu}) \end{aligned} \tag{4.1}$$

Begründung durch bestehende Produkte: Eine Entscheidung $decision(m_{neu})$ wird auf Grund einer oder mehrerer bereits existierenden Variablenbelegung $assignment(X_1 = x_{alt1}) \ldots assignment(X_n = x_{altM})$ getroffen (z.B. durch bestimmte Anforderungen aus dem Anforderungsdokument). Die Entscheidung soll ungültig werden, wenn mindestens eine der Wertzuweisungen ungültig wird. Um diesen logischen Zusammenhang formal zu beschreiben, erweitern wir Gleichung 3.2 um die Disjunktion der Negation der Prädikate $assignment(X_1 = x_{alt1})$ $\ldots assignment(X_M = x_{altM})$.

Die zusätzlichen Begründungen für die Gültigkeit einer Entscheidung verändern die Bedingungen für den Rückzug aus Gleichung 3.2 wie folgt:

$$\begin{aligned} &\neg assignment(I_i = i_{1k}) \vee \ldots \vee \neg assignment(I_n = i_{nk}) \vee \\ &\neg \mathbf{assignment(X_1 = x_{alt1})} \vee \ldots \vee \neg \mathbf{assignment(X_M = x_{altM})} \\ &\Rightarrow rejected - decision(m_{neu}) \end{aligned} \tag{4.2}$$

4.1.3 Begründung für den Rückzug von Entscheidungen

Begründung durch vorangegangene Entscheidungen: Eine Entscheidung $decision(m_{neu})$ wird zurückgezogen, weil eine oder mehrere zurückliegende Entscheidungen $decision(m_{alt1})$ ... $decision(m_{altM})$ getroffen wurden. Die Entscheidung $decision(m_{neu})$ soll ungültig werden, wenn mindestens eine der Entscheidungen $decision(m_{alt1})$ ... $decision(m_{altM})$ gültig wird. Um diesen logischen Zusammenhang formal zu beschreiben, wird Gleichung 3.2 um die Disjunktion der Prädikate $decision(m_{alt1}) \ldots decision(m_{altM})$ erweitert.

[2] Die **dick** markierten Prädikate in Gleichung 4.1 bis 4.4 stellen die zusätzlichen Begründungen dar.

Die zusätzlichen Begründungen für die Gültigkeit einer Entscheidung verändern die Bedingungen für den Rückzug aus Gleichung 3.2 wie folgt:

$$\begin{aligned}&\neg assignment(I_i = i_{1k}) \vee \ldots \vee \neg assignment(I_n = i_{nk}) \vee \\ &\mathbf{decision(m_{alt1})} \vee \ldots \vee \mathbf{decision(m_{altM})} \\ &\Rightarrow rejected - decision(m_{neu})\end{aligned} \tag{4.3}$$

Begründung durch bestehende Produkte: Eine Entscheidung $decision(m_{neu})$ wird zurückgezogen, weil eine oder mehrere bereits existierende Produkte $assignment(X_1 = x_{alt1}) \ldots assignment(X_n = x_{altM})$ (z.B. bestehende Anforderung) vorliegen. Die Entscheidung $decision(m_{neu})$ soll ungültig werden, wenn mindestens eine der Entscheidungen $assignment(X_1 = x_{alt1}) \ldots assignment(X_M = x_{altM})$ gültig wird. Um diesen logischen Zusammenhang formal zu beschreiben, erweitern wir Gleichung 3.2 um die Disjunktion der Prädikate $assignment(X_1 = x_1)..assignment(X_M = x_{altM})$.

Die zusätzlichen Begründungen für die Gültigkeit einer Entscheidung verändern die Bedingungen für den Rückzug aus Gleichung 3.2 wie folgt:

$$\begin{aligned}&\neg assignment(I_i = i_{1k}) \vee \ldots \vee \neg assignment(I_n = i_{nk}) \vee \\ &\mathbf{assignment(X_1 = x_{alt1})} \vee \ldots \vee \mathbf{assignment(X_M = x_{altM})} \\ &\Rightarrow rejected - decision(m_{neu})\end{aligned} \tag{4.4}$$

4.1.4 Einbindung der Erweiterungen in REDUX

Die beschriebenen zusätzlichen Begründungen wurden durch eine Erweiterung von REDUX in CoMo-Kit integriert.

Alle zusätzlichen Begründungen einer Entscheidung werden durch einen sogenannten **Design-Rationale-Knoten** verwaltet, der zu der entsprechenden Entscheidung gehört. Dieser Knoten wird eingeführt, um die Information in einer gemeinsamen Datenstruktur zu verwalten.

Die informelle textuelle Beschreibung einer Entscheidung wird in einem Slot dieses Knotens gespeichert. Dieser besteht selbst aus zwei Teilen:

- einer textuellen Beschreibung der Gründe für die Gültigkeit einer Entscheidung
- einer textuellen Beschreibung der Gründe für den Rückzug einer Entscheidung.

Die zusätzlichen Gründe für die Gültigkeit werden jeweils in die OUT-List einer Rechtfertigung des Design-Rationale-Knotens eingetragen. Dies gilt sowohl für Begründungen durch Produkte, die durch Assignment-Knoten dargestellt werden, als auch für Begründungen durch Entscheidungen, die durch Decision-Knoten repräsentiert werden.

Die zusätzlichen Gründe für den Rückzug werden jeweils in die IN-Liste einer Rechtfertigung des Design-Rationale-Knotens eingetragen. Dies gilt sowohl für Begründungen durch Produkte, die durch Assignment-Knoten dargestellt werden, als auch für Begründungen durch Entscheidungen, die durch Decision-Knoten repräsentiert werden.

Der Design-Rationale-Node selbst wird erst erzeugt sobald der Benutzer zusätzliche Begründungen eingibt. Er wird dann in die IN-Liste der Rechtfertigung des Rejected-Decision-Knotens eingetragen.

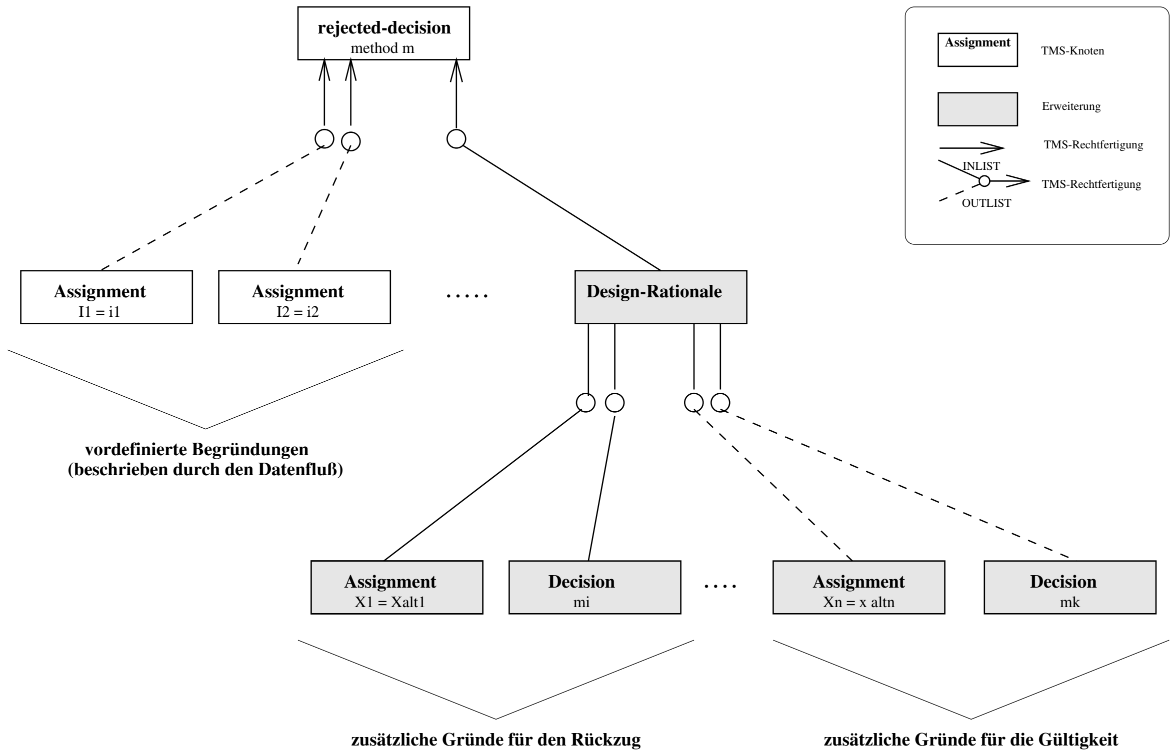

Abbildung 4.1: Abhängigkeiten zwischen den Begründungen und der Gültigkeit einer Entscheidungen

Es wird nun erläutert, wie sich Änderungen mit Hilfe der neuen Abhängigkeiten propagiert werden. Zu diesem Zweck sind zwei Übergänge interessant, die im folgenden beschrieben werden.

- **Eine Entscheidung ist gültig und eine der zusätzlichen Begründungen für die Gültigkeit wird ungültig:** Wird die zusätzliche Begründung für die Gültigkeit der Entscheidung ungültig, so wird der entsprechende Knoten mit OUT markiert. Dadurch bekommt der Design-Rationale-Knoten der Entscheidung eine gültige Rechtfertigung im Sinne des TMS und wird mit IN markiert.

Da der Design-Rationale-Knoten in der IN-List des Rejected-Decision-Knotens steht, wird dieser nun ebenfalls IN. Es besteht nun eine Notwendigkeit für den Rückzug der Entscheidung, und die Entscheidung wird ungültig.

- **Eine Entscheidung ist zurückgezogen und die Begründung für den Rückzug wird ungültig:** Für diese Betrachtung ist nur der Fall interessant, indem alle Gründe für die Gültigkeit gültig sind und nur ein Grund für den Rückzug gültig ist. Wird nun diese eine Begründung des Rückzuges ungültig, so wird der entsprechende Knoten von IN nach OUT markiert. Da dieser Knoten in der OUT-List einer Rechtfertigung steht, verliert der Design-Rationale-Knoten der Entscheidung die einzige gültige Rechtfertigung im Sinne des TMS und wird mit OUT markiert. Da der Design-Rationale-Knoten in der IN-List des Rejected-Decision-Knotens steht, wird dieser nun ebenfalls zu OUT. Es besteht nun keine Notwendigkeit mehr für den Rückzug der Entscheidung. Der Agent kann die Methode wieder auswählen.

Insgesamt legen wir für die Gültigkeit von Entscheidungen damit die folgende Semantik fest: Eine Entscheidung ist gültig, wenn alle Begründungen, die für die Entscheidung sprechen, gelten, und keine Begründung für den Rückzug zutrifft.

4.2 Einschränkung bestehender Begründungen

Begründungen von Entwurfsentscheidungen, die durch das Modell vorgegeben werden, beschreiben Datenflußabhängigkeiten. Sind durch das Modell Begründungen vorgegeben, die für die Entscheidung bedeutungslos sind, so ist es sinnvoll, sie während des Prozeßablaufes auf die tatsächlich existierenden Abhängigkeiten einzuschränken.

Dieser Schritt ist notwendig, um bei auftretenden Änderungen des Entwurfes nur auf die Prozeßschritte und Produkte zu verweisen, die tatsächlich von den Änderungen betroffen sind. Ein Beispiel verdeutlicht dies: Das Prozeßmodell gibt vor, daß das gesamte *Anforderungsdokument* in die Aufgabe *"Design"* eingeht. Ändert sich zu einem Zeitpunkt während der Designphase eine *nichtfunktionale Anforderung*, so ist es nicht nötig, das gesamte it Design zurückzunehmen, da manche dieser *Anforderung* sicher nur für den Entwurf einer der Komponenten eine Rolle spielten. Der Anwender hat zu diesem Zweck die Möglichkeit die Produkte zu selektieren, die keinen Einfluß auf die Entscheidung haben. Die Veränderung dieser nichtfunktionalen Anforderung wird dann nicht auf den Entwurf weiterpropagiert.

Die Anpassung der logischen Beschreibung aus Gleichung 3.2 wird erreicht, indem z.B. das Prädikat $assignment(I_n = i_{nk})$ aus der Formel herausgestrichen wird, das für die Entscheidung bedeutungslos ist. Die Gleichung 3.2 wird infolgedessen wie folgt angepaßt:

$$\begin{aligned}&\neg assignment(I_i = i_{1k}) \vee \ldots \vee \neg assignment(I_{n-1} = i_{n-1k})\\&\Rightarrow rejected - decision(m_{neu})\end{aligned} \tag{4.5}$$

Die Löschung von vorgegebenen Begründungen wird im TMS durch das Entfernen des entsprechenden Assignment-Knotens aus der Rechtfertigung des Rejected-Decision Knotens erreicht. Ist der Assignment-Knoten nicht mehr in der Rechtfertigung des Rejected-Decision-Knotens enthalten, so hat die Ungültigkeit diese Produktes keinen Einfluß mehr auf die Gültigkeit der Entscheidung.

Die durch Gleichung 4.1 bis 4.5 beschrieben Anpassungen schließen sich nicht gegenseitig aus, sondern können alle gleichzeitig als mögliche Erweiterungen bzw. Einschränkungen für eine Entscheidung durchgeführt werden.

Kapitel 5

Design Rationale in CoMo-Kit im Vergleich zu verwandten Arbeiten

Nachdem in Kapitel 2 einige Systeme vorgestellt wurden, die sich mit Entwurfsentscheidungen und deren Begründungen auseinander setzen, und in Kapitel 3 und 4 eine umfangreiche Darstellung von CoMo-Kit bezüglich dieser Thematik beschrieben wurde, soll dieses Kapitel nun einen zusammenfassenden Vergleich der Arbeiten darlegen. CoMo-Kit wird durch diese Darstellung im Vergleich zu den anderen Systemen bewertet. Auf einen Vergleich der Arbeiten untereinander wurde allerdings verzichtet, weil er den Rahmen dieser Arbeit überschreiten würde. Durch die Ausführungen soll deutlich werden, worin die prinzipiellen Unterschiede und Gemeinsamkeiten der verschiedenen Ansätze im Vergleich zu CoMo-Kit liegen. Dabei werden nicht alle Systeme aus Kapitel 2 aufgegriffen, sondern es werden anhand einiger Systeme die interessanten Aspekte herausgearbeitet.

5.1 CoMo-Kit im Vergleich zu IBIS

In Kapitel 3 wurde bereits beschrieben, wie die einzelnen Elemente der IBIS-Notation zu den CoMo-Kit-Elementen in Beziehung gesetzt werden können. Diese Darstellung hat verdeutlicht, daß die wesentlichen Elemente zur Darstellung einer Argumentation in den beiden Systemen übereinstimmen. Außerdem wurde in diesem Kapitel erläutert, welche Beziehungen und Begründungen in CoMo-Kit noch ergänzt wurden. Wir wollen nun abschließend noch einmal betrachten, welche Unterschiede und Gemeinsamkeiten das erweiterte CoMo-Kit im Vergleich zu IBIS aufweist.

- **Basisstrukturen**
  - **zur Darstellung der Argumentation:** Auch wenn die verschiedenen Elemente der Argumentation deutliche Gemeinsamkeiten aufweisen, so besteht doch ein wesentlicher Unterschied im Formalisierungsgrad der einzelnen Elemente. In IBIS wird die Information in *Issues*,*Positions* und *Arguments* durch beliebigen Text repräsentiert. In CoMo-Kit werden *Tasks*

und *Methoden* durch formale Elemente dargestellt, die im konzeptuellen Modell definiert sind. Diesen Elementen kann im konzeptuellen Modell eine textuelle Beschreibung zugeordnet werden. Begründungen können in CoMo-Kit ebenfalls durch formale Elemente oder durch einen Text beschrieben werden.

Betrachtet man die textuellen Beschreibungen der CoMo-Kit-Strukturen, so bauen diese ein IBIS-ähnliches Hypertextnetzwerk auf.

- **zur Darstellung von Produkt und Plan:** Da IBIS zur Darstellung beliebiger rhetorischer Argumentationen entwickelt wurde und sich nicht nur auf Entwurfsprozesse beschränkt, ist die Darstellung von Produkten und Plänen nicht vorgesehen. CoMo-Kit dagegen ist in der Lage, erzeugte Produkte in direktem Zusammenhang zur Argumentation darzustellen. Da Zwischenprodukte des Entwurfsprozesses den weiteren Entwurf häufig entscheidend beeinflussen, ist deren Referenzierung zur Begründung einer Entscheidung, wie es CoMo-Kit ermöglicht, besonders wichtig. Da die Produkte des Entwurfs ebenfalls durch CoMo-Kit verwaltet werden, muß der Anwender die Begründungen nicht in Form von Text eingeben, sondern kann direkt auf die verweisen.

- **Abhängigkeiten zwischen den Basisstrukturen:**

  - Die Abhängigkeiten zwischen CoMo-Kit-*Tasks* sind im Vergleich zu den Abhängigkeiten zwischen IBIS-*Issues* eingeschränkt. In CoMo-Kit bestehen sie aus den Abhängigkeiten durch die Aufgabenzerlegung. Dies entspricht der *serve*-Relation aus dem PHI-Dialekt von IBIS. Infolgedessen ist die Abhängigkeitsstruktur der CoMo-Kit-*Tasks* hierarchisch und damit wesentlich übersichtlicher, als die durch IBIS dargestellten Zusammenhänge zwischen *Issues*.

  - Im Unterschied zu IBIS kann man in CoMo-Kit den gegenseitigen Ausschluß von *Positions* (*Methoden*) darstellen. Dies gelingt, in dem man die Entscheidung für eine Methode als Grund für den Rückzug einer anderen Entscheidung angibt. In diesem Fall kann man eine *Methode* sowohl mit einem IBIS-*Argument* als auch mit einer IBIS-*Position* vergleichen.

- **Veränderung von Elementen und Beziehungen während der Prozeßabwicklung:** IBIS eignet sich besonders gut, um Sitzungsprotokolle zu dokumentieren. Der Anwender kann sehr einfach zusätzliche *Issues, Positions, Arguments* sowie deren Beziehungen ergänzen. Diese Eigenschaft von IBIS macht es besonders geeignet, um wenig bekannte Prozesse zu dokumentieren.

  In CoMo-Kit besteht eine Trennung der Information des Schedulers und des konzeptuellen Modells. *Tasks, Methoden* und *Produkte* werden im konzeptuellen Modell beschrieben. Auch die Abhängigkeiten zwischen Tasks und Methoden und den Produkten werden dort vorgegeben. Will der Anwender Methoden, Tasks oder Produkte ergänzen, so muß dies im konzeptuellen Modell gemacht werden. Die Information über den aktuellen Zustand der Aufgaben und die Wahl der Methoden ist im Scheduler enthalten. Hier werden auch die zusätzlichen Begründungen gespeichert und verwaltet. Sie können durch die beschriebenen Erweiterungen von CoMo-Kit komfortabel eingegeben werden. Will der Anwender während der Prozeßabwicklung Tasks oder Methoden ergänzen, so muß er zu diesem Zweck das konzeptuelle Modell ändern. Prinzipiell wäre es wünschenswert, dies auch im Scheduler zu ermöglichen, ohne das gesamte Prozeßmodell zu verändern. CoMo-Kit eignet sich besser, um Prozesse auszuführen, deren Modelle bekannt sind. In dieser Hinsicht unterscheidet sich CoMo-Kit deutlich von IBIS.

- **Computerunterstützung:**

  - **Entscheidungsunterstützung:** IBIS liefert keine Unterstützung bei der Auswertung von *Positions*, sondern beschränkt sich auf eine reine Darstellung der Argumentation. CoMo-Kit dagegen wertet Begründungen aus und bestimmt je nach deren Gültigkeit, ob die zugehörigen Entscheidung gültig ist oder nicht. Dabei wird die Auswertung der Argumentation durch die logische Implikation realisiert. Nur wenn alle Gründe für die Entscheidung zutreffen und kein Grund für deren Rückzug gilt, ist die Entscheidung gültig.

  - **Propagierung von Änderungen:** Da CoMo-Kit die Abhängigkeiten zwischen den Elementen der Argumentation verwaltet, kann die Wirkung von Änderungen über die Auswertung der Entscheidungen weiterpropagiert werden. Da IBIS Entscheidungen nicht auswerten kann, unterstützt es auch keine Propagierung von Änderungen.

  - **Graphische Darstellung:** In g-IBIS werden die Abhängigkeiten zwischen den Entscheidungen graphisch repräsentiert. CoMo-Kit liefert keine graphische Veranschaulichung der Zusammenhänge zwischen einzelnen Entscheidungen. Datenflußabhängigkeiten und Abhängigkeiten durch die Aufgabenzerlegung werden zwar mit Hilfe eines graphischen Editor dargestellt. Es ist allerdings nicht möglich den Zusammenhang der Entscheidungen graphisch sichtbar zu machen. Die im Rahmen dieser Diplomarbeit entwickelte Oberfläche ermöglicht es nur, einzelne Entscheidungen und deren Begründungen in textueller Form anzuzeigen.

  - **Generierung von Abhängigkeiten:** In CoMo-Kit wird im Gegensatz zu IBIS ein großer Teil der Abhängigkeiten aus der Information des konzeptuellen Modells automatisch generiert.

5.2 CoMo-Kit im Vergleich zu DRCS

DRCS stellt ebenso wie CoMo-Kit einen Formalismus zur Verfügung, der außer der Darstellung von Entwurfsentscheidungen auch die Beschreibung von Produkt und Plan sowie Abhängigkeiten zwischen diesen Elementen ermöglicht. Die Elemente der Repräsentationsform von DRCS zeigen erstaunliche parallelen zu den Strukturen von CoMo-Kit.

- **Basisstrukturen**

  Eine klare Zuordnung einzelner Elemente des Beschreibungsformalismus soll hier nicht vorgenommen werden, da DRCS eine sehr breite Palette unterschiedlicher Strukturen und Beziehungen beschreibt.

  - **zur Darstellung der Argumentation:** In DRCS stellt die Definition einer *Task*, eines *Modules*, einer *Spezifikation* und einer *Version* eine Entscheidung dar, die entsprechend argumentativ begründet wird. Die argumentative Begründung ist in CoMo-Kit nur für die Wahl von Methode vorgesehen. Da atomare Methoden aber Variablenbelegungen bestimmen und somit Produkteigenschaften festlegen, belegen Ihre Begründung die Wahl von Produkteigenschaften.
  - **zur Darstellung des Produktes:** DRCS beschreibt zwar ebenso wie CoMo-Kit auch das entwickelte (zu entwickelnde) Produkt, es bestehen jedoch Unterschiede in der Produktmodellierung. So setzt sich ein Produkt in DRCS aus *Modules, Interfaces* und *Connections* zusammen, während die Produkte in CoMo-Kit mittels *Konzepten* definiert werden. Die Aufteilung der Produkte in die drei oben genannten Teile, läßt sich in CoMo-Kit allerdings leicht durch eine Definition entsprechender Konzepte modellieren. *Constraints* können in CoMo-Kit im Gegensatz zu in DRCS nicht ausgedrückt werden.
  - **zur Darstellung des Plans:** DRCS beschreibt den Plan und dessen Zerlegung, ähnlich wie es in CoMo-Kit durch das konzeptuelle Modell erfolgt.

- **Abhängigkeiten zwischen den Basisstrukturen:**

  Die *has-plan*-Relation von DRCS zwischen Produktbeschreibung und Plan läßt sich mit der Abhängigkeit über den Datenflußgraph in CoMo-Kit vergleichen. Die Relation gibt beschreibt den Zusammenhang zwischen einem Produkt und der Aufgabe, die dieses Produkt generiert.

  Die Pläne werden ähnlich wie in CoMo-Kit durch eine Abfolge von Aufgaben beschrieben. Die Unterteilung in domain-level und meta-level einer Aufgabe, entspricht den atomaren und komplexen Aufgaben von CoMo-Kit.

  Ein *Decision-Problem* in DRCS, das durch die *the-best-option*-Relation beschrieben wird, entspricht einer gültigen Entscheidung in CoMo-Kit.

- **Computerunterstützung:**
  - **Entscheidungsunterstützung und Propagierung von Änderungen:** Die beschriebenen Beziehungen dienen in DRCS dazu, die Konsistenz der Lösung zu sichern. Ob DRCS eine automatische Auswertung der Entscheidung unterstützt und über die beschriebenen Abhängigkeiten somit eine Propagierung von Änderungen im Entwurf ermöglicht, wird in der Literatur nicht beschrieben.
  - **Generierung von Abhängigkeiten** Es ist in DRCS im Gegensatz zu CoMo-Kit nicht möglich Abhängigkeiten zwischen Produkt und Plan automatisch zu generieren, und den Anwender somit zu entlasten.

5.3 CoMo-Kit im Vergleich zu REMAP

Da REMAP die Basisstrukturen von IBIS verwendet, kann ein Vergleich dieser Elemente und deren Beziehungen an dieser Stelle unterbleiben. REMAP erweitert die IBIS Notation allerdings um eine Reihe von Strukturen und stellt damit ein Werkzeug zur Verfügung, das in seiner Funktionalität wesentlich mehr Übereinstimmungen zu CoMo-Kit zeigt als IBIS. Dennoch sind auch hier einige Unterschiede aufzuführen.

- **Basisstrukturen:**
  - **zur Darstellung der Argumentation:** REMAP verwendet *Assumptions*, um Annahmen des Entwurfsprozesses explizit von sicherem Wissen zu unterscheiden. CoMo-Kit stellt diese Art der Information nicht dar.
  - **zur Darstellung des Produktes:** In REMAP wird ähnlich wie in CoMo-Kit ein Zusammenhang zwischen Designentscheidungen und Designobjekten hergestellt. REMAP unterscheidet *Requirements* und *Design Objects* und nimmt damit eine Trennung der Anforderungen vom zu entwickelnden Produkt selbst vor. *Requirements* beeinflussen Entwurfsentscheidungen (*decisions*), dies entspricht in CoMo-Kit dem konsumieren von Information. *Design Objects* werden durch Entwurfsentscheidungen erstellt, dieser Zusammenhang entspricht den Wertzuweisungen durch atomare Methoden. In CoMo-Kit können *Requirements* als Produkte des Teilprozesses "Anforderungsentwicklung" beschreiben werden. Sie können also ebenfalls dargestellt werden, jedoch als Produkte des Entwurfsprozesses und nicht durch eine gesonderte Struktur.
  - **Darstellung des Planes:** REMAP sieht im Gegensatz zu CoMo-Kit keine Strukturen zur Darstellung des Planes vor.

- **Abhängigkeiten zwischen den Basisstrukturen:**

Die Beziehung zwischen *Requirements* und *Decisions* entspricht der Abhängigkeit zwischen der Begründung durch ein existierendes Produkt und einer Entscheidung in CoMo-Kit. Das existierende Produkt ist in diesem Fall ein *Anforderungsdokument*

Entscheidungen definieren in REMAP ähnlich wie in CoMo-Kit Zuweisungen an Produkte. In REMAP geschieht dies indirekt über die Festlegung von *Constraints*.

REMAP beschreibt durch die Relation *modifies*, in welcher Weise Entwurfsentscheidungen Anforderungen an das zu entwickelnde Produkt verändern. Diese Verbindung kann in CoMo-Kit nicht dargestellt werden. Wertvolle Information zur Entwurfsgeschichte des Produktes geht damit in CoMo-Kit verloren.

- **Computerunterstützung:**
  - **Entscheidungsunterstützung** REMAP wertet die Argumente aus, um die Gültigkeit einer Entscheidung zu bestimmen. REMAP verwendet dazu die gleiche Semantik wie CoMo-Kit.
  - **Propagierung von Änderungen:**
    In REMAP ist es ebenso wie in CoMo-Kit möglich, Änderungen im Entwurf zu propagieren. Der Anwender kann auf diese Weise erfahren, ob das *Design Object* bedingt durch die Änderungen im Entwurf seine Gültigkeit behält.
  - **Generierung von Abhängigkeiten:**
    CoMo-Kit generiert einen großen Teil der Abhängigkeiten automatisch aus den Abhängigkeiten, die im konzeptuellen Modell beschrieben sind. Der Anwender wird damit von einem Teil der zeitintensiven Dokumentation der Abhängigkeiten befreit. REMAP stellt diese Funktionalität nicht zur Verfügung.

5.4 CoMo-Kit im Vergleich zu REDUX

Da REDUX zur Verwaltung der Abhängigkeiten in CoMo-Kit eingesetzt und erweitert wurde, wird hier im wesentlichen erläutert, inwieweit die Funktionalität von CoMo-Kit über die von REDUX hinausreicht.

In REDUX werden keine Datenflußabhängigkeiten beschrieben. Infolgedessen werden sie auch nicht im Abhängigkeitsnetzwerk dargestellt. CoMo-Kit hingegen verwaltet diese Abhängigkeiten und kann sie zur Propagierung von Änderungen ausnutzten.

In REDUX beschreiben *Design rationales* die Gründe für die Optimalität einer Entscheidung. Werden diese Begründungen in REDUX ungültig, so wird die Entscheidung dadurch nicht invalidiert. In CoMo-Kit dagegen, wird die Entscheidung immer ungültig, wenn einer der Gründe für die Gültigkeit der Entscheidung entfällt. Sie sind in CoMo-Kit nicht an die Optimalität einer Entscheidung geknüpft. Die Optimalität einer Entscheidung wird in CoMo-Kit noch nicht betrachtet, da bisher noch keine Optimalitätsbedingungen angegeben wurden.

5.5 Zusammenfassung des Vergleiches

Die folgende Tabelle 5.1 soll im Überblick noch einmal verdeutlichen, welche wichtigen Funktionalitäten die beschriebenen Ansätze aufweisen. Dabei werden auch die Systeme aufgeführt, auf die in diesem Kapitel nicht eingegangen wurde. Die mit den Stern $\star$ gekennzeichneten Tabellenpositionen kennzeichnen, daß das System das beschriebene Merkmal besitzt. Ein Minus - bedeutet, das System besitzt da Merkmal nicht. Ein Fragezeichen drückt aus, daß in der Literatur bezüglich dieses Merkmals keine Aussage gefunden wurde.

Merkmale	CoMo-Kit	IBIS	REMAP	QOC	DRL	DRCS
Darstellung:						
der Argumentation:	⋆	⋆	⋆	⋆	⋆	⋆
des Produktes	⋆	-	⋆	⋆	-	⋆
des Plans	⋆	-	-	-	-	⋆
Entwurfsgeschichte	-	⋆	⋆	-	?	⋆
Constraints	-	-	⋆	-	-	⋆
Auswertung von Argumenten	⋆	-	⋆	-	⋆	?
Automatische Generierung von Abhängigkeiten	⋆	-	-	-	-	-
Propagierung von Änderungen	⋆	-	⋆	-	⋆	?

Tabelle 5.1: Vergleich der beschriebenen Ansätze anhand einiger Merkmale

Durch die vergleichende Analyse werden bereits einige Aspekte deutlich, die in CoMo-Kit zusätzlich integriert werden könnten. Das Kapitel 7 wird einige dieser Ideen noch einmal aufgreifen, um Hinweise auf mögliche Erweiterungen von CoMo-Kit aufzuzeigen.

Kapitel 6

Beispiel

In den folgenden Ausführungen wird ein Beispiel die beschriebenen Erweiterungen von CoMo-Kit verdeutlichen.

Eine einführende Beschreibung der Beispieldomäne soll das Umfeld des Beispiels erläutern. Anschließend werden die Basisstrukturen von CoMo-Kit beschrieben. Die folgenden Abschnitte stellen die Ergebnisse dieser Arbeit anhand des Beispieles dar.

6.1 Die Beispieldomäne

Da das Beispiel aus der Anwendungsbereich "Softwareentwicklung" gewählt wurde, sollte der Leser mit den Begriffen dieses Domäne vertraut sein.

Das Beispiel lehnt sich an den Softwareentwicklungsprozeß des SFB-501 dar. Im Rahmen des Sonderforschungsbereiches wird ein Kontrollsystem zur Gebäudeautomatisation entwickelt. Ziele und Begriffe des Projektes *Gebäudeautomatisation* werden hier in kürze erläutert.

Es soll ein System entwickelt werden, mit dessen Hilfe die Haustecknik eines Gebäudekomplexes kontrolliert und gesteuert werden soll. Zu diesem Zweck muš die Beleuchtung, das Klima und die Zugangskontrolle geregelt werden. Wir betrachten im Folgenden zwei Elemente dieses Systems etwas genauer.

- Die Sensoren dienen der Erfaßung der Meßwerte. Dazu gehören zum Beispiel Thermometer, Feuchtigkeitsmeßgeräte und Bewegungsmelder.

- Die Kontrollzellen beschreiben die Räume, deren Kontrolle von Licht, Klima, Zugangskontolle usw. durch das entwickelte Kontrollsystem gesteuert werden soll.

6.2 Das konzeptuelle Modell

Mit Hilfe von CoMo-Kit wird der Entwurfsprozeß in Form eines *konzeptuellen Modelles* beschrieben. Dieses wird bei der Durchführung instantiiert.

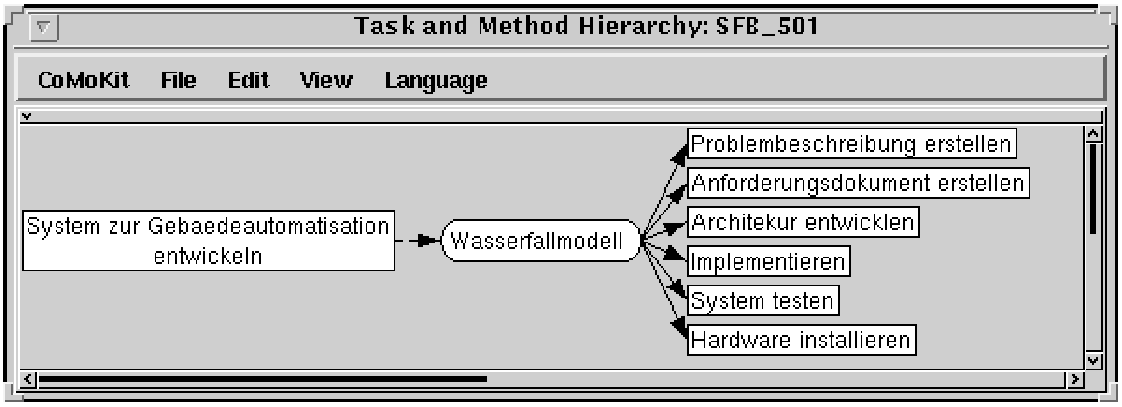

Abbildung 6.1: Konzeptuelles Modell in CoMo-Kit

In Abbildung 6.1 und 6.2 sind Teile des konzeptuelle Modell dargestellt.

Aufgabenzerlegung: Die Task-Methoden-Hierarchie (Abb. 6.1) beschreibt den Zusammenhang der Basisstrukturen *Tasks* und *Methoden*, indem sie die Aufgabenzerlegung vorgibt. Die Task *System zur Gebäudeautomatisation entwickeln* kann durch Anwendung der komplexen Methode *Wasserfallmodell* bearbeitet werden. Diese definiert wiederum neue Teilaufgaben: *Problembeschreibung erstellen, Anforderungsdokument erstellen, Architektur entwickeln, ...* .

Datenfluß: In Abbildung 6.2 wird der Datenfluß für die Methode *Wasserfallmodell* beschrieben. Die Rechtecke repräsentieren, die Tasks, die auch in Abb. 6.1 sichtbar sind. Die Ovale stellen die *formalen Parameter* dar. Die Abbildung macht deutlich, daß die *Problembeschreibung* aus der Aufgabe *Problembeschreibung erstellen* hervorgeht und zur Bearbeitung von *Anforderung erstellen* benötigt wird. Außerdem kann man in der Darstellung des Datenfluß erkennen, daß die Aufgaben *Implementieren* und *Hardware installieren* parallel durchgeführt werden können, sobald die *Architekturbeschreibung* vorliegt. Beide Aufgaben tragen zur Erstellung des *Gesamtsystems* bei.

Im Folgenden wird das Modell noch verfeinert. In Abbildung 6.3 ist zu erkennen, daß zur Bearbeitung der Aufgabe *Architektur entwickeln* zwei alternative Methoden zur Verfügung stehen. Die Architekur kann von einer einzigen Gruppe entwicklet werden (*keine Aufteilung in Gruppen*) oder die Entwicklung erfolgt in zwei unabhängigen Teilgruppen (*Aufteilung in Gruppen*), von denen jede einen Teil der Architekur definiert. Der Anwender wählt bei der Instantiierung des Modelles eine dieser Methoden. Diese Wahl repräsentiert eine *Entscheidung* für die eine oder andere Methode.

Betrachten wir die Methode *Aufteilung in Gruppen* etwas genauer. Die Datenflußabhängigkeiten dieser Methode sind in Abbildung 6.4 zu sehen. Sie gliedert sich in die Teilaufgaben *Sensoren entwerfen, Kontrollzellen entwerfen* und *Schnittstelle beschreiben*, die nach der Vorgabe des Modelles parallel durch geführt werden können.

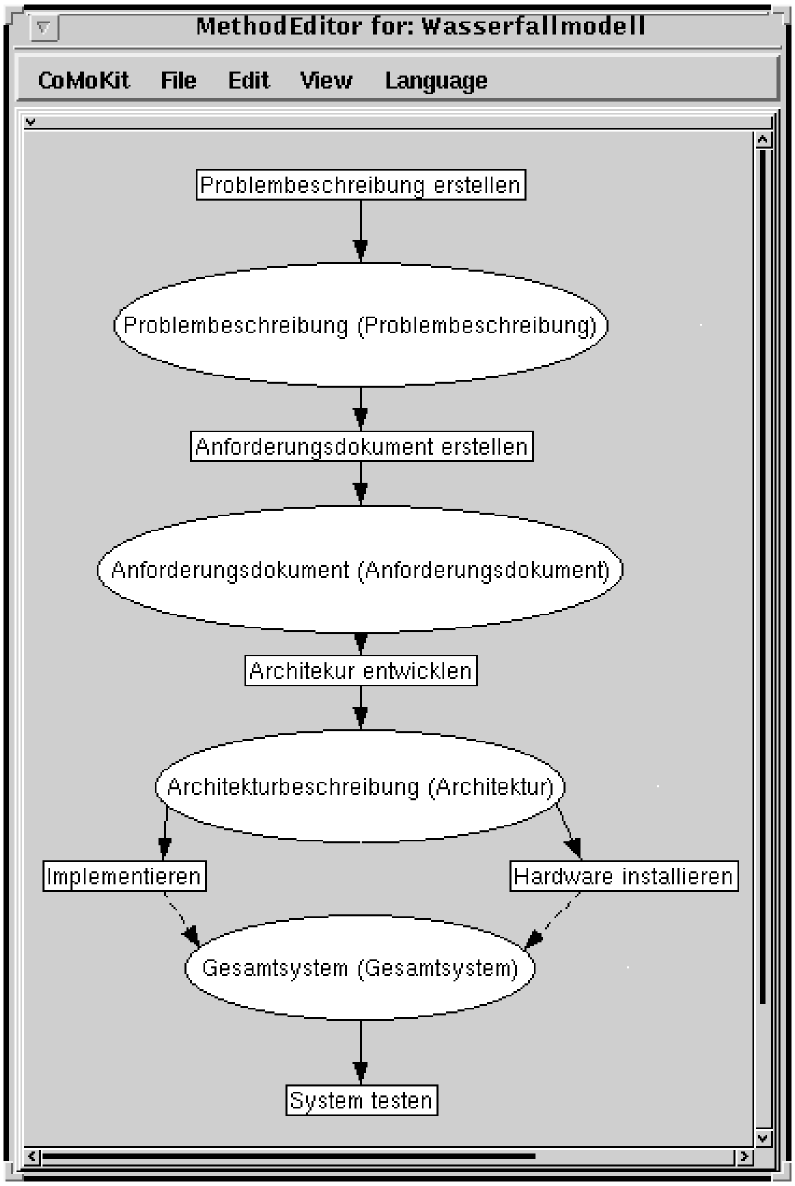

Abbildung 6.2: Datenfluß der Methode *Wasserfallmodell*

Die Durchführung des Projektes nach dem beschriebenen Prozeßmodell sah wie folgt aus: Zur Entwicklung eines Systems zur Gebäudeautomatisation wurde die Wasserfallmethode verwendet. Nachdem die Problembeschreibung erstellt war, und auf dieser Grundlage das Anforderungsdokument entwickelt wurde, beschloß man das Entwicklungsteam in zwei Gruppen aufzuteilen. Diese sollten jeweils einen Teil der Architektur getrennt voneinander entwerfen. Gruppe 1 soll die Architektur der Sensoren beschreiben, Gruppe 2 soll die Architekur der Kontrollzellen entwerfen.

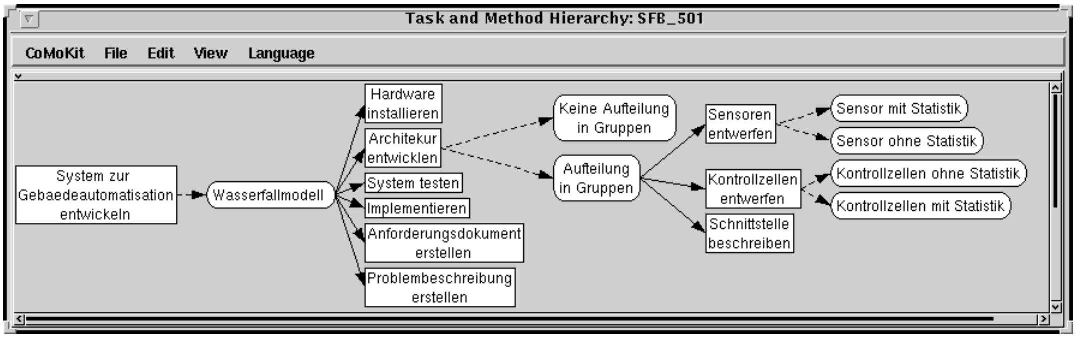

Abbildung 6.3: Verfeinerung des konzeptuelles Modells

Nachdem nun die prinzipielle Darstellung der Prozeßbeschreibung mittels des konzeptuellen Modelles erläutert wurde, soll nun beschrieben werden, wie die Ergebnisse dieser Arbeit zur Unterstützung des Entwurfsprozeßes beitragen.

6.3 Darstellung zusätzlicher Begründungen

Beim Entwurf der Kontrollzellen- und Sensorarchitektur werden eine Reihe von Entscheidungen getroffen, die letztendlich die Gesamtarchitekur bestimmen. In diesem Beispiel wollen wir eine dieser Entscheidungen genauer betrachten, um mit Hilfe dieser Entscheidung die Ergebnisse der Arbeit zu verdeutlichen.

Im *Anforderungsdokument* ist beschrieben, daß das Gesamtsystem eine Statistik der Meßwerte führen soll, in der zum Beispiel die Außentemperatur im Verlaufe eines Tages aufgezeichnet ist. Das *Anforderungsdokument* rechtfertigt die Entscheidungen bei der Entwicklung der Architektur. Das Dokument ist als Begründung für diese Entscheidungen bereits durch das konzeptuelle Modell vorgegeben.

Beim Entwurf der Architekur der Kontrollzellen stellte sich nun die Frage, ob diese Statistik im Sensor oder in der Kontrollzelle geführt werden soll.

Im konzeptuellen Modell von Abb. 6.3 stehen zwei Methoden zur Auswahl *Kontrollzelle mit Statistik* und *Kontrollzelle ohne Statistik*. Gruppe 1 entscheidet sich dafür, die Statisik nicht in der Kontrollzelle zu führen. Diese Entscheidung wird durch die Gültigkeit des Prädikat $decision(Kontrollzelle\ \ ohne\ \ Statistik)$ ausgedrückt.

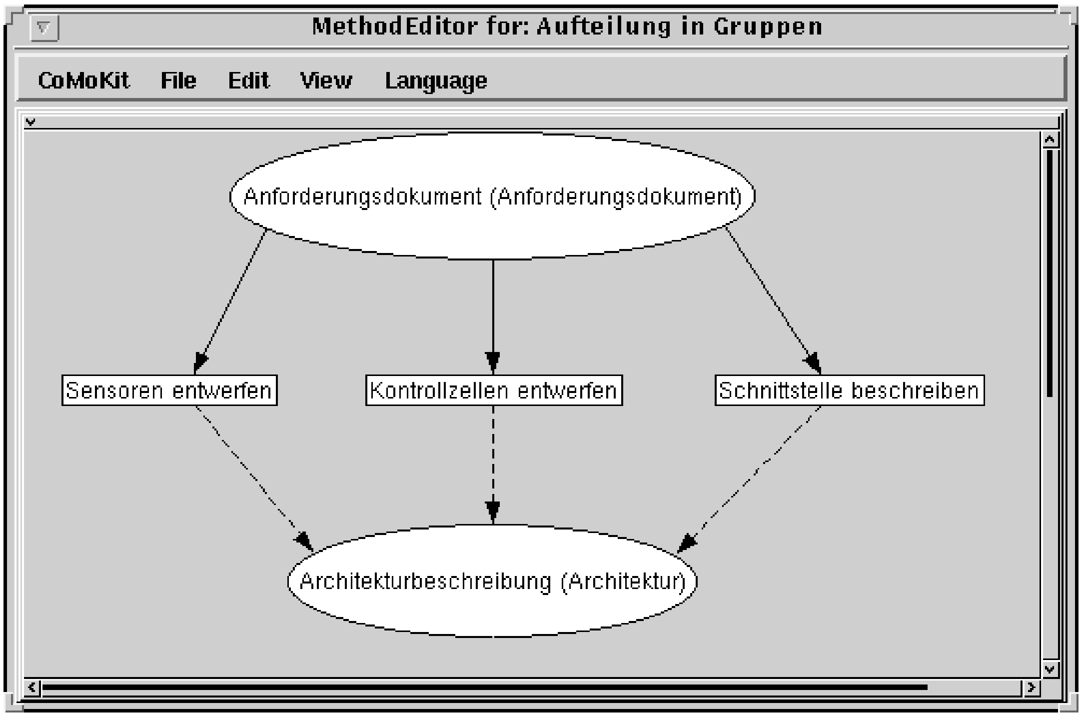

Abbildung 6.4: Datenflußgraph für die Methode *Aufteilung in Gruppen*

Da für diese Entscheidung bereits eine Begründung (nämlich das *Anforderungsdokument*) durch den Datenfluß definiert ist, beschreiben wir den Zusammenhang aus Gleichung 3.2 (Abschnitt 3.3.2) wie folgt:

$$\begin{aligned} &\neg assignment(I_1 = Anforderungsdokument) \\ &\Rightarrow rejected\_decision(Kontrollzelle\quad ohne\quad Statistik) \end{aligned} \tag{6.1}$$

Verschiedene Überlegungen haben zu dieser Entscheidung geführt. Sie beschreiben zusätzliche Begründungen, die in CoMo-Kit dokumentiert werden können:

- Zum einen hatte Gruppe 1 beim Entwurf der Sensoren bereits die Entscheidung *Sensoren mit Statistik* getroffen. Diese Begründung beruht also auf einer vorangegangenen Entscheidung.

- Außerdem, ist die Belastung des Netzes bei *Kontrollzellen ohne Statistik* geringer. Denn werden die Statistikdaten in den Sensoren selbst gespeichert, so muß nicht jede ermittelte Temperatur zur Kontrollzelle übertragen werden. Nur dann, wenn diese an einem bestimmten Statistikwert interessiert ist, kann sie dem Sensor dies über eine entsprechende Anfrage mitteilen und der Wert wird übertagen. Diese Begründung soll in Form einer textuellen Begründung eingegeben werden.

Die Darstellung der Begründungen wird folgendermaßen erreicht. Der Anwender, in diesem Fall der *Teamleader von Gruppe 2*, dokumentiert die Überlegungen mit Hilfe der Oberfläche, die im Rahmen dieser Arbeit entstanden ist. Durch die Selektion eines bestimmten Menüpunktes oder Buttons im Design Rationale Fenster, auf das wir gleich zurückkommen werden, teilt der Anwender mit, daß er seine Entscheidung durch eine anderen Entscheidung begründen will (in unserem Beispiel wird die Entscheidung *Kontrollzelle ohne Statistik* mit Hilfe der Entscheidung *Sensor mit Statistik* begründet). Zur Beschreibung der Begründung öffnet sich ein Dialog. Dieser ist in Abbildung 6.5 zu sehen. Alle bisher getroffenen Entscheidungen, die als Begründung herangezogen werden dürfen, sind in diesem Fenster dargestellt. Sie sind auf der linken Seite des Dialogfensters zu sehen. Aus dieser Liste kann man nun alle Entscheidungen selektieren, die als Begründung dienen. Die ausgewählten Entscheidungen werden in eine Liste auf die rechte Seite verschoben. Hat der Anwender alle Begründungen gewählt, so schließt er den Dialog mit *Accept*. Die neue Information wird dann ins Abhängigkeitsnetzwerk des Schedulers übernommen. Der logische Zusammenhang aus Gleichung 6.1, wird darauf hin um die zusätzliche Begründung erweitert:

$$\neg assignment(I_1 = Anforderungsdokument) \vee \neg decision(Sensor\ \ mit\ \ Statisitk) \\ \Rightarrow rejected\_decision(Kontrollzelle\ \ ohne\ \ Statistik) \tag{6.2}$$

Die neue Begründung wird im Design Rationale Window (Abb. 6.6) ergänzt. In diesem Fenster sieht man auf der linken Seite in der Liste *Operators of all Decisions* alle getroffenen Entscheidungen. Selektiert man eine der Entscheidung, so werden alle Begründungen dieser Entscheidung im Fenster sichtbar. Im dargestellten Fenster ist die Entscheidung *Kontrollzelle ohne Statistik* angewählt. Zu dieser Entscheidung ist die folgende Information sichtbar:

- Unter *Predefined Parameters* stehen die Begründungen, die durch den Datenfluß vorgegeben sind. In diesem Fall ist das das Anforderungsdokument, das bei der Entwicklung der Architektur, die getroffenen Entscheidungen begründet. Im Anforderungsdokument ist zum Beispiel beschrieben, das die Statistik geführt werden soll.

- In der Liste *Additional Parameters* werden zusätzliche Begründungen durch bestehende Produkte beschrieben. Begründungen dieser Art wurden hier nicht angegeben.

- Die Liste *Previous Decisons* zeigt die neu eingegebenen Begründung an.

- Die Überlegungen bezüglich der Netzbelastung werden vom Anwender direkt im unteren Feld *Textual Justification* eingegeben.

- Zur Entscheidungen sind keine Begründungen des Rückzuges dargestellt.

- Außerdem zeigt das Fenster noch an, wer die Entscheidung getroffen hat, die hier begründet wurde (hier: *Teamleader Gruppe 2*, und welche Alternative zur getroffenen Entscheidung möglich gewesen wäre.

Die Buttons die am oberen Rand des Fensters angebracht sind, dienen dazu entspechende Aktionen mit der selektierten Entscheidung durchzuführen. Eine Beschreibung der Funktionen ist der Systemdokumentation im Anhang zu entnehmen.

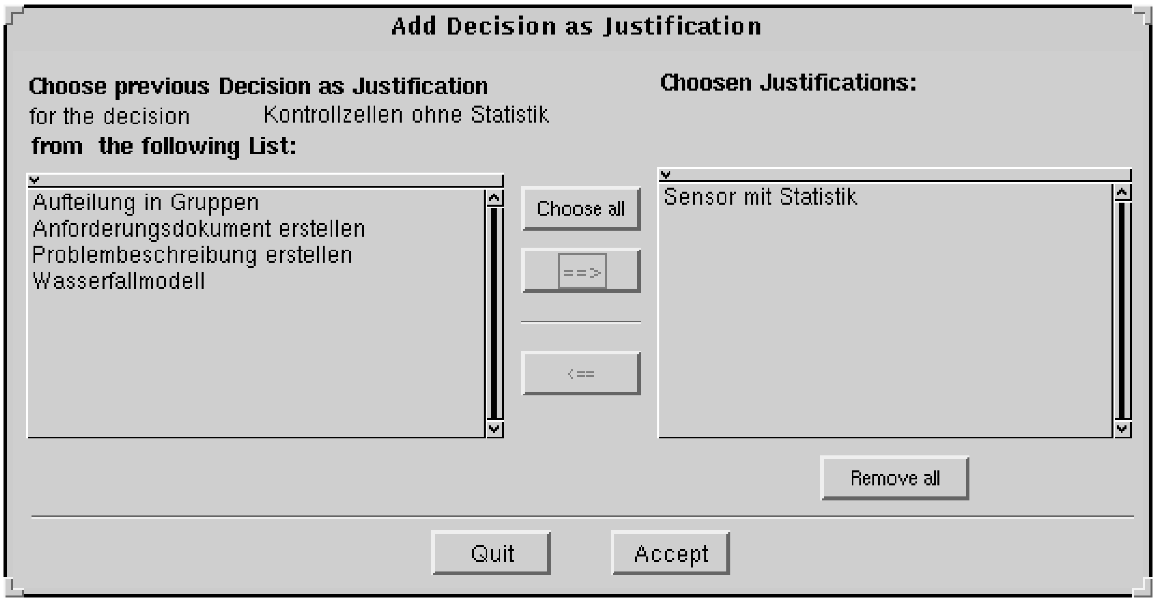

Abbildung 6.5: Dialog zur Eingabe der zusätzlichen Begründung

6.4 Propagierung von Veränderungen

Nachdem nun beschrieben wurde, wie die Begründungen dargestellt werden, um sie für den weiteren Entwurf zu speichern, wollen wir nun erläutern wie die Abhängigkeiten die durch diese Begründungen aufgebaut wurden, den weiteren Entwicklungsprozeß unterstützen können.

Zu einem späteren Zeitpunkt des Entwurfes wird die Entscheidung *Sensoren mit Statisik* zurückgezogen. Gruppe 1 beschlieẞt dies, weil die Architktur durch die Verwaltung der Statistikdaten so komplex wird, und die Anforderungen an die Speicherelemente der Sensoren zu kostspielig sind. Die Entscheidung *Sensor mit Statistik* wird ungültig. Damit wird die Begründung der Entscheidung *Kontrollzellen ohne Statistik* hinfällig. Der Zusammenhang aus Gleichung 6.2 macht dies deutlich. Das Prädikat $rejected\_decision(Kontrollzelle\ \ ohne\ \ Statistik)$ wird gültig. Die Notwendigkeit des Rückzuges bewirkt, daß die Entscheidung $decison(Kontrollzelle\ \ ohne\ \ Statistik)$ ungültig wird.

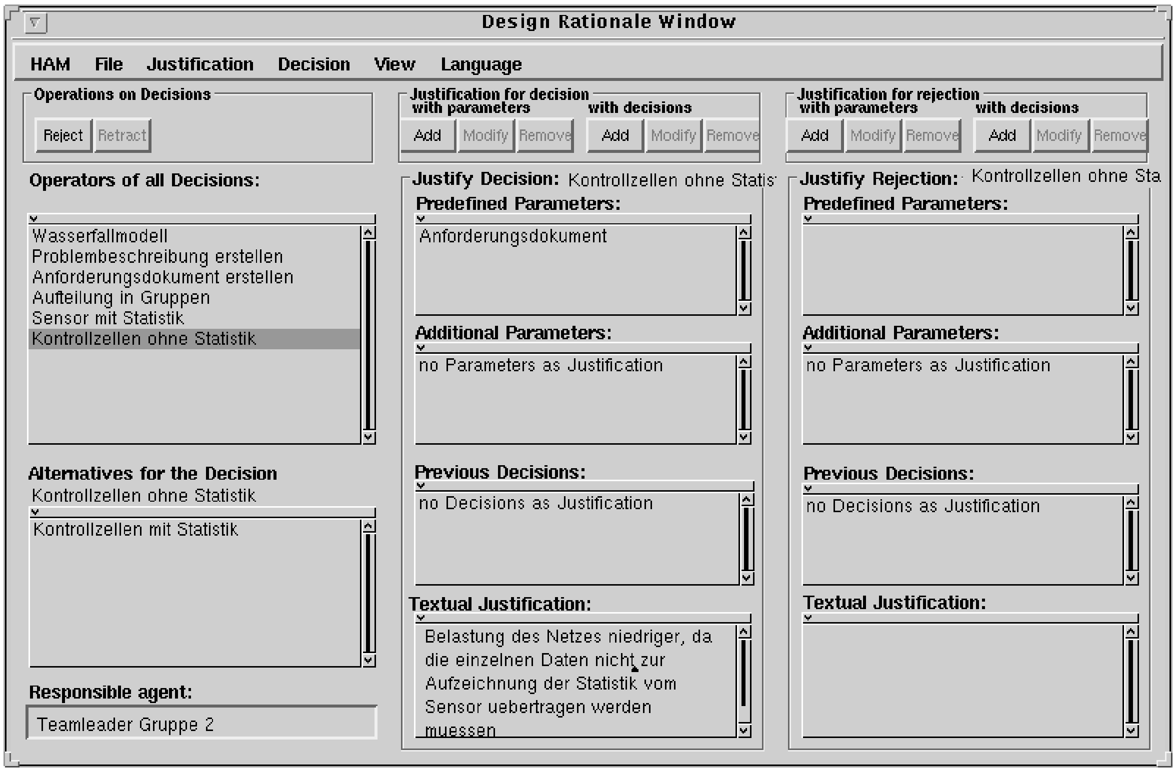

Abbildung 6.6: Das Design Rationale Window: Fenster zur Darstellung der Entscheidungen und Begründungen

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel faßt die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf mögliche Entwicklungen im Zusammenhang mit CoMo-Kit und Design Ratinale.

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurden Erweiterungen an CoMo-Kit vorgenommen, die es ermöglichen, Begründungen von Entwurfsentscheidungen während der Prozeßabwicklung zu beschreiben.

Im bisherigen System wurden die Abhängigkeiten aus der Information des konzeptuellen Modells generiert. Es hat sich gezeigt, daß diese Abhängigkeiten den aktuellen Entwicklungsprozeß nicht genau genug beschreiben, da die Vorgaben aus dem konzeptuellen Modell nicht immer optimal auf die konkrete Projektsituation zugeschnitten sind oder die notwendige Information erst bei der Ausführung des Planes zur Verfügung steht. Durch Erweiterungen an CoMo-Kit im Rahmen dieser Arbeit, kann der Anwender nun auch während der Ausführung Begründung seiner Entwurfsentscheidungen ergänzen. Er kann die Begründungen sowohl durch Eingabe einer textuellen Beschreibung, als auch durch Referenzierung existierender Produkte oder getroffener Entscheidungen angeben. Bei der Begründung durch Produkte oder Entscheidungen, werden zusätzliche Abhängigkeiten zwischen den entsprechenden Elementen aufgebaut. Gibt das konzeptuelle Modell zu viele Begründungen vor, so kann der Anwender diese bei der Ausführung des Planes entfernen. Beide Veränderungen tragen dazu bei, nur die Abhängigkeiten zu verwalten, die für das Projekt von Bedeutung sind.

Da die Abhängigkeiten in CoMo-Kit nicht nur dargestellt werden, sondern mittels REDUX auch verwaltet werden, können Effekte von Entscheidungen propagiert werden. Nur wenn das Abhängigkeitsnetzwerk auch die konkrete Projektsituation widerspiegelt, kann dieser Mechanismus die Projektabwicklung auch optimal unterstützen.

Die in Abschnitt 2.2 beschriebenen Erwartungen an diese Arbeit, konnten mit den Ergebnissen erfüllt werden. Die Darstellung der Begründungen wird ermöglicht und unterstützt das **Verständnis des Entwurfes**. Aufbauend auf der bereits integrierten Abhängigkeitsverwaltung kann somit eine **computerunterstützte Propagierung von Änderungen** erfolgen.

Zusammenfassend kann man sagen, daß CoMo-Kit, die in der Einleitung beschriebene Problematik bei der Planung, Steuerung und Abwicklung komplexer Entwurfsprozessen, auf folgende Weise unterstützt:

- Entscheidungen und deren Begründungen werden in einem Abhängigkeitsnetzwerk dokumentiert. Auf diese Weise wird Wissen gespeichert, das sonst nur implizit in den Entwurfsprozeß eingeht.

- Die Client-Server-Architektur ermöglicht die Koordination großer Entwicklungsteams. Die Entscheidungen der einzelnen Entwickler werden in einem gemeinsamen Abhängigkeitsnetzwerk dargestellt. Jeder Entwickler kann sich die Entscheidungen der anderen Mitarbeiter anhand dieser Aufzeichnungen verdeutlichen.

- Die TMS-basierte Abhängigkeitsnetzwerk verwaltet den komplexen Zusammenhang aus Entscheidungen und Begründungen und ist in der Lage, die Information zur Umplanung auszunutzen. Durch die Propagierung von Veränderungen über das Netzwerk, können betroffene Mitarbeiter von einer notwendigen Überarbeitung in Kenntis gesetzt werden.

7.2 Ausblick

Insbesondere aus der Betrachtung der in Kapitel 2 beschriebenen Ansätze und deren Vergleich mit CoMo-Kit in Kapitel 5, haben sich die folgenden Ideen entwickelt. Sie beschreiben, welche Weiterentwicklungen in CoMo-Kit denkbar wären.

7.2.1 Auswertung der Argumente

Die Auswertung der Argumente (Begründungen) in CoMo-Kit wird durch eine logische Implikation realisiert. Es müssen alle Begründungen, die für eine Entscheidung sprechen, erfüllt sein, damit eine Entscheidung gültig ist. Gleichzeitig, darf kein Argument, das gegen die Entscheidung spricht, zutreffen.

Diese Auswertungsstrategie unterscheidet sich deutlich vom Prozeß der Entscheidungsfindung eines Menschen, bei dem Begründungen gegeneinander abgewägt werden, bevor eine Entscheidung getroffen wird. Wollte man diesen Prozeß modellieren, so müßte man einzelnen Argumenten verschiedene Gewichte zuordnen und eine Auswertungsstrategie zur Verrechnung dieser Gewichte beschreiben (Stichwort: Fuzzy-Logik). Erst dann wäre es möglich, Effekte von Entscheidungen und Begründungen über Abhängigkeiten zwischen diesen zu propagieren.

Eine Bewertung der Argumente mit Gewichten aus einem großen Wertebereich erscheint nicht sinnvoll realisierbar, besonders in Hinblick auf die notwendige Auswertungsstrategie. Man könnte sich aber vorstellen, den Begründungen Attribute der Form *"notwendig"* und *"unbestimmt"* zuzuordnen. Eine mögliche Auswertung der Entscheidung könnte dann folgendermaßen aussehen: Damit eine Entscheidung gültig ist, müssen alle *notwendigen* Begründungen erfüllt sein. Sind *unbestimmte* Begründungen nicht erfüllt, so wird der Anwender davon informiert, und trifft eine Entscheidung über die Bedeutung dieses Argumentes. Er legt damit fest, welchen Einfluß dieses Argument auf die Gültigkeit der Entscheidung im aktuellen Zusammenhang hat. Diese Realisierung entspricht einer dreiwertigen Logik, in der eine Entscheidung den Zustand *gültig, ungültig* und *unbestimmt* annehmen kann. Sie erinnert an die in Abschnitt 2.3.8 beschriebene Logik des issuebasierten TMS.

In CoMo-Kit könnte solch ein Verhalten realisiert werden, in dem die *unbestimmte* Begründung an einen Optimalitätsverlust gekoppelt würde. Diesen haben wir in den Ausführungen von Abschnitt 2.3.7 beschrieben. In REDUX stellen die *Design Rationales* Optimalitätsbedingungen dar. Werden sie ungültig, so wird die Entscheidung nicht zurückgezogen, sondern der Anwender wird davon in Kenntnis gesetzt.

Man sollte prüfen, ob die hier beschriebene Auswertung von Entscheidungen nicht mit einem großen Aufwand für den Entwickler verbunden wäre, da er bei einer Umplanung unter Umständen sehr viele Entscheidungen noch einmal einzeln prüfen müßte.

7.2.2 Weiter Anpassungen bei der Ausführung eines Planes

Durch die in dieser Arbeit beschriebenen Erweiterungen von CoMo-Kit ist es möglich Begründungen während der Ausführung eines Planes zu ergänzen und anzupassen. Diese Information wird nicht im konzeptuellen Modell dargestellt, sondern nur im aktuellen Abhängigkeitsnetzwerk des Schedulers. Man unterscheidet damit zwei Sorten von Information. Information, die nur für das aktuelle Projekt wichtig ist und Information, die für alle Projekte, die das Modell verwenden, von Bedeutung ist.

Für Methoden und Produkte können wir diese Unterscheidung bisher noch nicht treffen. Will der Anwender während der Ausführung ein Ziel, eine Methode oder ein Produkt ergänzen oder verändern, so muß er dies im konzeptuellen Modell tun. Häufig beziehen sich aber auch diese Veränderungen nur auf die konkrete Projektsituation und sollen nicht prinzipiell ins Modell übernommen werden. Aus diesem Grund wäre es sinnvoll, auch Aufgaben und Methoden während der Prozeßabwicklung einfügen zu können, ohne das konzeptuelle Modell zu verändern, Ähnlich, wie dies bereits für Begründungen möglich ist.

7.2.3 Graphische Darstellung der Entscheidungen

Im Rahmen dieser Arbeit wurde eine Oberfläche entwickelt, die dem Anwender alle Informationen bezüglich der getroffenen Entscheidungen zur Verfügung stellt. Das

Fenster zeigt einzelne Entscheidungen, deren Begründungen und Alternativen, und gibt an, welcher Agent die Entscheidung getroffen hat.

Es wäre schön, wenn sich der Anwender den Zusammenhang der einzelnen Entscheidungen zusätzlich mit Hilfe einer graphischen Oberfläche verdeutlichen könnte. In dieser sollten die Entscheidungen und Begründungen in Form von Knoten repräsentiert werden und deren Abhängigkeiten über entsprechende Kanten beschrieben werden. Die Darstellung könnte in Anlehnung an g-IBIS erfolgen (vergleiche Abschnitt 2.3.2).

7.2.4 Fragen zum Entwurf

Bei der Entwicklung eines Produktes wäre es hilfreich, wenn CoMo-Kit Fragen vom folgenden Typ beantworten könnte:

- Welche Entscheidungen wurden bei der Bearbeitung einer bestimmten Aufgabe getroffen?
- Durch welche Entscheidungen wurde eine bestimmte Produktcharakteristik bestimmt?
- Welche Aufgaben hat ein bestimmter Agent bearbeitet?
- Für welche Entscheidungen ist ein bestimmtes Produkt von Bedeutung?
- Welche Entscheidungen wurden bereits zurückgezogen?

Antworten auf diese oder ähnliche Fragen zum Entwurf können den Anwender im Laufe des weiteren Entwurfes unterstützen, weil sie ihm wichtige Fragen über die Zusammenhänge und Hintergründe von Entscheidungen beschreiben. Der Anwender selbst hat Schwierigkeiten die Information aus der Komplexität der Zusammenhänge herauszufiltern.

Die Information zur Bearbeitung der Anfragen ist im Abhängigkeitsnetzwerk von CoMo-Kit enthalten. Um Antworten zu generieren, müßte allerdings eine geeignete Anfragesprache, sowie ein Mechanismus zum effizienten Zugriff der entsprechenden Information entwickelt werden.

Arango et al. [ABCF91] haben eine Entwicklungsumgebung beschrieben, die Anfragen an den Entwurf bearbeitet.

7.2.5 Darstellung der Entwurfsgeschichte

In CoMo-Kit werden Veränderungen des bestehenden Entwurfes durch deren Propagierung unterstützt. CoMo-Kit dokumentiert jedoch nicht, in welcher zeitlichen Reihenfolge die einzelnen Entscheidungen getroffen und verworfen wurden. Diese Information könnte dazu verwendet werden, sich anhand der Entscheidungen die Entwurfsgeschichte des Produktes zu verdeutlichen. Eine Analyse der Entwurfsgeschichte kann die Erstellung von Prozeßmodellen erleichtern und damit zur Entwicklung "guter" Prozeßmodelle beitragen.

7.2.6 Konfliktvermeidung

In CoMo-Kit kann ein Agent nur die Teilaufgaben bearbeiten, die ihm aufgetragen wurden. Auf diese Weise stellt das System sicher, daß Konflikte bei der Bearbeitung vermieden werden.

Den Rückzug einer Entscheidung kann jeder Agent vornehmen, unabhängig davon, ob er oder ein anderer Agent die Entscheidung getroffen hat. Ebenso kann jeder Agent Begründungen für Entscheidungen oder deren Rückzug eingeben, ohne daß kontrolliert wird, ob dieser Agent die Entscheidung auch getroffen hat. Um Konflikte zu vermeiden, wäre es sinnvoll, den Entscheidungen Verantwortlichkeiten zuzuordnent, die eine Zugriffskontrolle sichern.

Literaturverzeichnis

[ABCF91] A. Arango, L. Bruneau, J.-F. Cloarec, and A. Feroldi. A tool shell for tracking design decisions. *IEEE Software*, pages 75–82, March 1991.

[AID92] *Working Notes of AAAI'92 Workshop on Design Rationale Capture and Use*, American Association of Artificial Intelligence, San Jose, CA, July 1992.

[Bel93] V. Bellotti. Integating theoreticians' and practioners' perspectives with design rationale. In *Proceedings of the 1993 Conference on Human Factors in Computing Systems, INTERACT'93 and CHI'93*, Amsterdam, April 1993.

[CR91] J.M. Carroll and M.B. Rosson. Deliberated evolution: Stalking the view matcher in design space. *Human-Computer Interaction*, 6:281–318, 1991.

[CY91] E.J. Conklin and B. Yakemovic. A process-oriented approach to design rationale. *Human-Computer Interaction*, 6:357–391, 1991.

[Del94] Barbara Dellen. Verwaltung von Abhängigkeiten bei der Operationalisierung konzeptueller Modelle. Diplomarbeit, Universität Kaiserslautern, 1994.

[DJ88] Vasant Dhar and Matthias Jarke. Dependency directed reasoning and learning in systems maintenance support. In *IEEE Transactions on Software Engineering* [Lub91], pages 211–227.

[DMP95] B. Dellen, F. Maurer, and J. Paulokat. Verwaltung von Abhängigkeiten in kooperativen wissensbasierten Arbeitsabläufen. In M. Richter and F. Maurer, editors, *Proceedings in Artificial Intelligence 2, Expertensysteme 95*, 1995.

[Doy79] J. Doyle. A truth maintenance system. *Journal on Artificial Intelligence*, 12(3):231–272, 1979.

[FLMM91] G. Fischer, A.C. Lemke, R. McCall, and A.I. Morch. Making argumentation serve design. *Human-Computer Interaction*, pages 393–419, 1991.

[FMM89] Gerhard Fischer, Raymond McCall, and Anders Morch. JANUS: Integrating hypertext with Knowledge-based design environment. In *Hypertext '89 Proceedings*, pages 105–117, November 1989.

[GR91] Thomas R. Gruber and Daniel M. Russell. Design knowledge and design rationale : a framework for representation, capture and use. Technical Report KSL 90-45, Knowledge System Laboratorys, Stanford University California 94305, August 1991.

[GR92] Thomas R. Gruber and Daniel M. Russell. Derivation and use of design rationale information as expessed by designers. Technical Report KSL 92-64, Knowledge Systems Laboratory, July 1992.

[Gru90] Thomas R. Gruber. Model-based explanation of design rationale. In *Proceedings of the AAAI-90: Explanation Workshop*, July 1990.

[HA93] Masaki Hamada and Hisato Adachi. Recording software design processes for maintaining the software. In *Procceeding: IEEE Computer Society's International Computer Software and Applications Conference - COMPSAC 93*, pages 27–33, Los Alamitos, CA, USA, 1993. Computer Society Press.

[JLS92] Alex P. J. Jarczyk, Peter Löffler, and Frank M. Shipman. Design rationale for software engineering: a survey. In B. et al. Milutinovic, V.; Shriver, editor, *Proceedings of the 25th Hawaii Internationale Conference on System Science*, pages 577–586, Los Almitos, Ca, USA, January 1992.

[Kle93] Mark Klein. Capturing design rationale in concurrent engineering teams. *IEEE Computer*, pages 39–47, January 1993.

[Kle94] Mark Klein. iDCSS: Integrating workflow, conflict and rationale-based concurrent engineering coordination technologies. In *Concurrent Engineering: Research and Applications , Conference 1994*, pages 189–202, 1994.

[KR70] W. Kunz and H. Rittel. Issues as elements of information systems. Working paper no.131, Institute of Urban and Regional Development, University of California, Berkley, 1970.

[Lee90] Jintae Lee. Sibyl: a tool for managing group decision rationale. In *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 79–92, Los Angeles, CA, October 1990.

[LL91] Jintae Lee and Kum-Yew Lai. What's in design rationale? *Human-Computer Interaction*, 6:251–280, 1991.

[Lub91] Mitchell D. Lubars. Representing design dependencies in an issue-based style. In *IEEE Software* [GR92], pages 81–88.

[Mau93] Frank Maurer. *Hypermediabasiertes Knowledge-Engineering für verteilte wissensbasierte Systeme*. Dissertation, Universität Kaiserslautern, 1993.

[McC86] Raymond J. McCall. Issue-serve systems: A descriptive theory of design. *Design Methods and Theories*, 20(8):443–458, 1986.

[MM94] Diane McKerlie and Allan MacLean. Reasoning with design rationale: practical experience with design space analysis. *Design Studies*, 15(2):214–225, April 1994.

[MP95] Frank Maurer and Gerhard Pews. Ein Knowledge-Engineering-Ansatz für kooperatives Design am Beispiel Bebauungsplanung. *Künstliche Intelligenz, Themenheft Knowledge Engineering*, (1):28–34, January 1995.

[MYBM91] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. Questions, options and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6:201–250, 1991.

[PB88] Colin Potts and Glenn Bruns. Recording the reasons for design decisions. In *Proceedings of the 10th International Conference on Software Engineering*, pages 418–427, 1988.

[PCP94] C. Petrie, M. Cutkosky, and H. Park. Design space naviagtion as collaborative aid. In J.S. Gero and F. Sudweeks, editors, *Artificial Intelligence in Design '94*, pages 611–623. Kluwer Academic Publishers, Netherlands, 1994.

[Pet91] Charles Petrie. *Planning and Replanning with reason maintenance*. PhD thesis, MCC AI Lab, Austin, TX, 1991.

[PL92] F. Pena and R. Logcher. Capturing design assumptions for rational coordination, integration and negotation in the design process. *Computing System in Engineering*, 3(6):651–660, 1992.

[RD94] Balasubramaniam Ramesh and Vasant Dhar. Representing and maintaining process knowledge for large-scale systems development. *IEEE Expert*, 9(2):54–59, April 1994. +.

[Rit93] Helmut Ritzer. Konzeption und Implementierung einer TMS-basierten Komponente zur Verwaltung von Abhängigkeiten bei der Konfiguration und Planung. Diplomarbeit, Universität Kaiserslautern, 1993.

[Sch94] Willi-Gerd Schmitz. Multiple Aufgabenzerlegung von konzeptuellen Modellen. Diplomarbeit, Universität Kaiserslautern, 1994.

[Sch95] Frank Schepp. Verwaltung von Abhängigkeiten in flexiblen Arbeitsabläufen. Diplomarbeit, Universität Kaiserslautern, Juli 1995.

[SH94] Simon Buckingham Shum and Nick Hammond. Argumentation-based design rationale: what use at what cost? *International Journal of Human-Computer Studies*, 40:603–652, 1994.

[Sys] Corporate Memory Systems. Cm/1 product description. 8920 Business Park Dr., Austin, TX 78759, USA.

[Tou58] Stephen E. Toulmin. *The use of argument.* Cambridge Universtiy Press, Cambridge, England, 1958.

[YC90] Burgess Yakemovic and Jeffrey Conklin. Report on a development project use of an issue-based information system. In *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW)*, Los Angeles, CA, October 1990.

[Zac86] Wayne Zachary. A cognitively based functional taxonomy of decision support techniques. *Human-Computer Interaction*, 2:25–63, 1986.