# Agentenbasierte Konsistenzprüfung heterogener Modelle in der Automatisierungstechnik

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Michael Rauscher
aus Backnang

| | |
|---|---|
| Hauptberichter: | Prof. Dr.-Ing. Dr. h. c. Peter Göhner |
| Mitberichter: | Prof. Dr.-Ing. Birgit Vogel-Heuser |
| Tag der Einreichung: | 09.06.2015 |
| Tag der mündlichen Prüfung: | 28.09.2015 |

Institut für Automatisierungs- und Softwaretechnik
der Universität Stuttgart

2015

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart.

Mein besonderer Dank an dieser Stelle gilt meinem Doktorvater Herrn Prof. Dr.-Ing. Dr. h.c. Peter Göhner, für die Unterstützung, die kritischen Diskussionen sowie für die vielen wertvollen Hinweise und Ideen, die zum Gelingen dieser Arbeit beigetragen haben.

Ebenso möchte ich mich bei Frau Prof. Dr.-Ing. Birgit Vogel-Heuser für das Interesse an meiner Arbeit und für die Übernahme des Mitberichts bedanken.

Herzlich möchte ich mich bei meinen ehemaligen Kollegen am IAS für die hervorragende Zusammenarbeit und die Unterstützung während meiner Zeit am Institut bedanken. Die Diskussionen haben viel zu dieser Arbeit beigetragen, genauso wie die gegenseitige Motivation. Ohne sie wäre die Zeit am Institut nur halb so schön gewesen. Nicht vergessen möchte ich an dieser Stelle die Kollegen von anderen Instituten und Universitäten, mit denen ich beispielsweise in Forschungsprojekten zusammengearbeitet habe.

Ein ebenfalls gebührender Dank gilt den Studierenden, die durch ihre Arbeiten zum Gelingen dieser Arbeit beigetragen haben. An viele unterhaltsame und motivierende Momente erinnere ich mich gerne zurück.

Zu guter Letzt, möchte ich mich bei meiner Familie und meinen Freunden für die Unterstützung, den Rückhalt und das mir entgegengebrachte Verständnis bedanken.

Stuttgart, im Oktober 2015

Michael Rauscher

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| AMS | **A**gent **M**anagement **S**ystem |
| AOSE | **A**gent **O**riented **S**oftware **E**ngineering |
| BDI | **B**elieve **D**esire **I**ntention |
| CAD | **C**omputer **A**ided **D**esign |
| CAx | **C**omputer **A**ided **x** |
| DF | **D**irectory **F**acilitator |
| DL | **D**escription **L**ogic |
| EMF | **E**clipse **M**odeling **F**ramework |
| FIPA | **F**oundation for **I**ntelligent **P**hysical **A**gents |
| GMF | **G**raphical **M**odeling **F**ramework |
| GUI | **G**raphical **U**ser **I**nterface |
| ID | **Id**entifier |
| JADE | **J**ava **A**gent **De**veloping Framework |
| MAS | **M**ulti-**a**gent **S**ystem (dt.: Agentensystem) |
| MaSE | **M**ulti-**a**gent **S**ystems **E**ngineering |
| MOF | **M**eta **O**bject **F**acility |
| OO | **O**bjekt**o**rientierung |
| OWA | **O**pen **W**orld **A**ssumption |
| OWL | **W**eb **O**ntology **L**anguage [W3C12] |
| PASSI | **P**rocess for **A**gent **S**ocieties **S**pecification and **I**mplementation |
| RDF | **R**esource **D**escription **F**ramework |
| SPARQL | **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage |
| SPS | **S**peicher**p**rogrammierbare **S**teuerung |
| SQL | **S**tructured **Q**uery **L**anguage |
| TGG | **T**riple **G**raph **G**rammatik |
| UML | **U**nified **M**odeling **L**anguage |
| W3C | **W**orld **W**ide **W**eb **C**onsortium |

| | |
|---|---|
| WBS | **W**issens**b**asiertes **S**ystem |
| XML | E**x**tensible **M**arkup **L**anguage |

# Begriffsverzeichnis

**Agent:** Ein Agent ist eine autonome (Software-) Einheit, die selbstständig Ziele verfolgt und die in der Lage ist, mit ihrer Umgebung und anderen Agenten zu kommunizieren.

**Agentensystem:** Ein System aus mehreren Agenten wird als Agentensystem bezeichnet. Im Englischen wird der Begriff „multi-agent system“ (MAS) verwendet.

**Aspekt:** Ein Aspekt bezeichnet eine Betrachtungsweise auf einen Sachverhalt. Je nach Kontext kann ein Aspekt als eine disziplinspezifische Sicht oder eine bestimmte Thematik verstanden werden.

**Begriff (Ontologie):** Ein Begriff bezeichnet innerhalb einer Ontologie eine Klasse, also einen Knoten des in der Ontologie enthaltenen Graphen.

**Blackboard:** Blackboard ins Deutsche übersetzt bedeutet „(Schreib-)Tafel“. Das Blackboard-Verfahren bezeichnet ein verteiltes Problemlösungsverfahren mit gemeinsamer Datenhaltung. Alle beteiligten Parteien benutzen ihr jeweiliges Wissen und ändern bzw. ergänzen die Daten so lange, bis ein für alle befriedigendes Ergebnis vorliegt.

**Elementarfunktion:** Funktionen können so lange zerlegt und verfeinert werden, bis sie nur noch aus Elementarfunktionen bestehen. Elementarfunktionen oder Grundfunktionen bezeichnen somit die einfachsten Grundoperationen wie „leiten“, „wandeln“, „mischen“ oder „trennen“. [Koll98]

**Entität:** Eine Entität bezeichnet einen konkreten oder abstrakten Gegenstand und dient als Sammelbegriff für Dinge, Sachverhalte, Relationen, Eigenschaften oder Ereignisse.

**Frühe Entwurfsphase:** Die frühe Entwurfsphase oder der frühe Entwurf bezeichnet im Rahmen dieser Arbeit die gemeinsame, disziplinübergreifende Entwurfsphase der Mechatronik. Sie findet vor der späten, disziplinspezifischen Entwurfsphase statt. Ihr Ergebnis ist die Prinziplösung bzw. das Konzept für das zu entwickelnde Produkt.

**Funktion:** Eine Tätigkeit, die ein System durchführt, wird als Funktion bezeichnet. Funktionen werden mit Verben oder substantivierten Verben beschrieben. Beispiele sind (Kraft) „erzeugen“ oder (Produkt) „produzieren“. [Koll98]

**Globale Begriffswelt:** Die globale Begriffswelt definiert die allgemeinen, systemweit geltenden Begriffe und deren Bedeutung. Auf der globalen Begriffswelt bauen unter anderem die Konsistenzregeln auf. Lokale Begriffe müssen deshalb in globale Begriffe übersetzt werden können, um ein Modell zu interpretieren.

**Inkonsistenz:** Mit Inkonsistenz werden im Rahmen dieser Arbeit Fehler oder Widersprüche in Modellen bezeichnet. Inkonsistenzen können innerhalb eines einzelnen Modells oder zwischen mehreren Modellen auftreten.

**Lokale Begriffswelt:** Unter der lokalen Begriffswelt werden die Begriffe eines bestimmten Modells bzw. dessen Modellierungssprache verstanden. Sie sind spezifisch und gelten häufig nur für dieses (lokale) Modell.

**Mechatronik:** Die Mechatronik ist eine Kombination der Disziplinen Mechanik, Elektrotechnik und Informationstechnik. Je nach Definition können auch weitere Disziplinen wie die Regelungstechnik oder die Fluidtechnik dazugerechnet werden.

**Meta-Modell:** Das Meta-Modell beschreibt die Struktur eines Modells und beschreibt so den Modelltyp. Es definiert die verfügbaren Modellelemente und deren mögliche Verknüpfungen. Das Meta-Modell ist vergleichbar mit einem Klassendiagramm, das die Struktur eines zugehörigen Objektdiagramms festlegt.

**Modell:** Ein Modell ist ein Abbild einer Vorstellung (präskriptiv/vorschreibend) oder der Realität (deskriptiv/beschreibend). Dabei ist das Abbild in der Regel nicht allumfassend sondern beschreibt Ausschnitte bzw. verschiedene Aspekte des Ganzen. In dieser Arbeit werden die verschiedenen Entwurfsartefakte einer Entwicklung als Modell bezeichnet.

**Modellelement:** Die einzelnen Elemente eines Modells werden als Modellelemente bezeichnet. Im Fall eines Klassendiagramms sind das beispielsweise die Klassen, deren Attribute und Methoden sowie die Assoziationen und Kardinalitäten.

**Modelltyp:** Der Begriff Modelltyp wird verwendet, wenn ausgedrückt werden soll, dass es sich nicht um ein konkretes Modell, sondern um einen Typ von Modell (z. B. Klassendiagramme) handelt.

**Ontologie:** Eine Ontologie ist eine formale Darstellung einer Menge von Begriffen und die zwischen ihnen bestehenden Beziehungen sowie Axiome, die Zusammenhänge beschreiben, die nicht durch Begriffe und Beziehungen ausgedrückt werden können.

**Partialmodell:** Ein Partialmodell bezeichnet ein Modell, das Teil eines Ganzen ist. Das Ganze wird mit Hilfe von mehreren Modellen beschrieben, die in der Regel unterschiedliche Aspekte beschreiben. Hier ist ein Modell des Systems der Partialmodelle nach [GFDK09] gemeint.

**Prinziplösung:** Die Prinziplösung stellt das technische Konzept für ein zu entwickelndes Produkt dar. Sie enthält die Ideen zur Lösung der in den Anforderungen spezifizierten Aufgaben und ist das Ergebnis der frühen Entwurfsphase. Sie wird in der Regel disziplinübergreifend erarbeitet und dient als Referenz für die weitere disziplinspezifische Entwicklung.

**System von Modellen:** Wenn mehrere Modelle gemeinsam zum Beispiel ein Produkt beschreiben, spricht man von einem System von Modellen. Jedes Modell beschreibt einen anderen Blickwinkel auf das Produkt bzw. einen anderen Aspekt. Da sie jedoch dasselbe Produkt beschreiben, besitzen sie in der Regel viele Abhängigkeiten zueinander und bilden deshalb ein System von Modellen.

**Wissensbasierte Systeme:** Wissensbasierte Systeme sind Systeme, die Wissen enthalten, welches zwischen allgemeinen Problemlösungswissen und spezifischem Fachwissen getrennt ist. [RFSE11]

# Zusammenfassung

Automatisierte bzw. mechatronische Systeme sind aus der heutigen Welt nicht mehr wegzudenken. Sowohl ein Großteil der technischen Produkte als auch die industrielle Produktion enthalten Elemente der Mechatronik. Sie ist eine Disziplin, die die klassischen Disziplinen des Maschinenbaus, der Elektrotechnik und der Informationstechnik in sich vereint. Um die gewinnbringende Zusammenarbeit dieser sehr unterschiedlichen Disziplinen zu gewährleisten, muss verschiedenen Herausforderungen begegnet werden. Die durch die verschiedenen Disziplinen bedingte Verwendung heterogener Modelle ist dabei eine große Fehlerquelle beim Entwurf mechatronischer Systeme. Inkonsistenzen zwischen den einzelnen Modellen, die erst spät im Entwurfsprozess aufgedeckt werden, erfordern großen Aufwand zur Behebung und gefährden die Qualität der entwickelten Produkte. Auf Grund der Heterogenität der Modelle, ist eine automatisierte Prüfung der Modelle auf Inkonsistenzen nicht möglich. Die manuelle Abstimmung zwischen den Disziplinen und Modellen kosten die Entwickler jedoch viel Zeit im Entwicklungsprozess und sie können sich nicht vollständig auf ihre eigentliche, kreative und gewinnbringende Arbeit konzentrieren. Deshalb ist die Unterstützung der Entwickler bei der Prüfung der Modelle notwendig, um die Entwicklungskosten zu reduzieren und die Produktqualität zu steigern.

In der vorliegenden Arbeit wird ein Konzept zur automatisierten Konsistenzprüfung für heterogene Modelle vorgestellt, das in der Lage ist, Inkonsistenzen aufzudecken und den Entwicklern alle zu deren Auflösung verfügbaren Informationen bereitzustellen. Das Konzept basiert darauf, die heterogenen Modelle und deren Inhalt auf eine globale, modellübergreifende Ebene zu abstrahieren und dort zu interpretieren und zu prüfen. Dies wird erreicht, indem jedes Modell, genauer jeder Modelltyp, eine (lokale) Beschreibung erhält, in der die Struktur und die Bedeutung der Syntax des Modelltyps enthalten sind. Das zur Prüfung erforderliche Wissen in Form von Fakten, Zusammenhängen und daraus abgeleiteten Regeln ist allgemeingültig verfasst und befindet sich auf der globalen Ebene. Benutzerspezifisch können zu prüfende Regeln ergänzt werden. Die lokalen und die globale Wissensbasen werden in Form von Ontologien realisiert. Die Prüfung selbst wird mit Hilfe eines Softwareagentensystems durchgeführt. Die Grundidee der agentenorientierten Konsistenzprüfung ist es, jedes beteiligte Modell mit einem Modellagenten zu repräsentieren, der als Schnittstelle zwischen lokaler und globaler Ebene fungiert. Ein Ontologieagent leitet die Regeln aus der globalen Wissensbasis ab. Regelagenten führen die Prüfung der Modelle auf die Einhaltung der Regeln aus. Ein Koordinierungsagent stößt die Prüfung an, koordiniert diese und verwaltet die Ergebnisse. Dem Entwickler werden diese Ergebnisse am Ende einer Prüfung präsentiert und alle Informationen, die zu einem positiven oder negativen Ergebnis geführt haben, bereitgestellt. Er kann die aufgedeckten Inkonsistenzen anschließend gezielt beheben und wird so bei seiner Tätigkeit unterstützt. Gleichzeitig wird die Qualität der mechatronischen Systeme gesteigert, indem die möglichen Fehlerquellen reduziert werden.

# Abstract

It is not possibly to imagine today's world without mechatronic and automated systems. Both, a big part of the technical products and the industrial production consist of mechatronic elements. Mechatronic is a discipline which combines the classical disciplines of mechanical engineering, electrical engineering and information technology. In order to ensure the gainful collaboration of these very diverse disciplines, it is necessary to meet with some challenges. The use of heterogeneous models, caused by the different involved disciplines, is a big source of error for the development of mechatronic systems. Inconsistencies between the single models, which are discovered late in the development process, require a big effort to solve them and endanger the quality of the product. Due to the heterogeneity of the models, an automated check of the models for inconsistencies is not possible. The developers have to spend much time for the manual coordination of the disciplines and the models and cannot concentrate completely on their classical creative and gainful work. Due to that fact, the support of the developers checking the models is necessary, in order to reduce the costs of the development and to improve the quality of the product.

This thesis proposes a concept for the automated consistency check of heterogeneous models, which is able to detect inconsistencies and to provide the developers with the available information to solve them. The idea of the concept is to abstract the heterogeneous models and their content to a global, model comprehensive layer and to interpret them on that layer. This is achieved by adding a (local) description to each model respectively model type, which contains the structure and the meaning of the syntax of the model type. The knowledge, which is needed for the consistency check in form of facts, relations and derived consistency rules, is composed generally and arranged on the global layer. User defined consistency rules can be added. The local and the global knowledge base are realized in form of ontologies. The consistency check itself is provided by a software agent system. The basic idea of the agent-oriented consistency check is to represent each involved model by a model agent. He has the function of an interface between the local and the global layer. An ontology agent derives the consistency rules from the global knowledge. Rule agents check if the rules are fulfilled by the models. A coordination agent initiates and coordinates the check and manages the results. At the end of the check the results are presented to the developer including all information which led to a positive or negative result. Afterwards, he is able to solve the found inconsistencies systematically and by that he is supported during his work. Simultaneously, the quality of the mechatronic systems is improved by reducing possible error sources.

# 1 Einleitung und Motivation

Die Anforderungen an heutige technische Produkte sind in den letzten Jahren ständig gewachsen. Gewachsen sind jedoch auch die zu deren Realisierung zur Verfügung stehenden Technologien und damit das Wissen der Ingenieure. Dieses Wissen hat einen solchen Umfang angenommen, dass es von einer einzelnen Person nicht mehr erfasst werden kann. Viele Produkte werden deshalb heute von Teams entwickelt, deren Mitglieder Experten auf unterschiedlichen Gebieten sind. Diese unterschiedlichen Gebiete werden auch als Disziplinen bezeichnet und sind beispielsweise der Maschinenbau, die Elektrotechnik oder die Informationstechnik. Sie sind entstanden, um das Wissen zu strukturieren und damit besser beherrschbar zu machen. Weitere Erkenntnisgewinne erfordern weitere Spezialisierungsschritte und es entstehen neue (Teil-) Disziplinen wie beispielsweise die Feinwerktechnik, die Fluidmechanik oder die Starkstromtechnik.

## 1.1 Multidisziplinäre Zusammenarbeit

Um qualitativ hochwertige Produkte, die den hohen Anforderungen gerecht werden, zu entwickeln, muss deshalb das Wissen der verschiedenen Disziplinen während der Entwicklung wieder vereint werden. Dies wird erreicht, indem Ingenieure unterschiedlicher Disziplinen in einem sogenannten multidisziplinären Team zusammenarbeiten. So können die Technologien aller beteiligten Disziplinen genutzt werden, was neue Möglichkeiten bietet, neuartige und optimierte Produkte zu entwickeln.

Die Mechatronik ist ein typisches Beispiel für die multidisziplinäre Zusammenarbeit und wird als Begriff oft synonym zur Automatisierungstechnik gebraucht. Da der Begriff der Mechatronik die Multidisziplinarität mehr heraushebt, wird dieser Begriff im weiteren Verlauf verwendet. Er bezeichnet das Zusammenspiel des Maschinenbaus, der Elektrotechnik und der Informationstechnik. Funktioniert die Zusammenarbeit gut, ist die Mechatronik mehr als die Summe der Einzeldisziplinen. Bei Entwicklungen sind die einzelnen Disziplinen häufig unterschiedlich stark vertreten und je nach Produkt muss ein anderer Schwerpunkt gesetzt werden.

Zur Entwicklung eines mechatronischen Systems, beispielsweise eines Automobils, wird Wissen aus fast allen Bereichen und Disziplinen benötigt. Deshalb ist es notwendig, dass Ingenieure aus unterschiedlichen Fachrichtungen zusammenarbeiten. Aus jedem Bereich werden die grundlegenden, aber auch die neuesten Erkenntnisse miteinbezogen. So ist gewährleistet, dass das Wissen aller beteiligten Bereiche angewendet wird und die jeweiligen Vorteile genutzt werden. Erst durch diese multidisziplinäre Zusammenarbeit kann ein zuverlässiges, sicheres, komfortables und umweltfreundliches Automobil entstehen.

Das Zusammenspiel von mehreren Disziplinen tritt jedoch auch in vielen weiteren Bereichen auf. Im Bauwesen arbeiten beispielsweise Architekten und Statiker zusammen. Zur Entwicklung eines Flugzeugs sind zunächst Strukturmechaniker, Aerodynamiker und Flugmechaniker gefragt. Für das Innenleben werden dann zusätzlich zum Beispiel Elektrotechniker oder weitere Disziplinen benötigt. Bei der Entwicklung einer Fertigungsstraße sind Maschinenbauer, Elektrotechniker, Informationstechniker, Fertigungsplaner usw. beteiligt. Es gibt also fast keine Entwicklung mehr, bei der nicht mehrere Disziplinen zusammenarbeiten müssen. Die meisten Entwicklungen bzw. Produkte würden ohne diese Zusammenarbeit nicht existieren.

## 1.2 Herausforderungen multidisziplinärer Zusammenarbeit

So groß der Gewinn aus multidisziplinärer Zusammenarbeit auch sein mag, so gibt es doch auch gewisse Herausforderungen, denen begegnet werden muss.

Die erste Herausforderung lässt sich einfach erkennen: Das Wissen wurde auf mehrere Disziplinen aufgeteilt, damit es beherrschbar bleibt. Wenn dieses Wissen wieder vereint wird, können zwar Synergien entstehen, jedoch steigt in der Regel auch die Komplexität. Das Wissen unterschiedlicher Bereiche bzw. Disziplinen muss berücksichtigt werden, wobei die Disziplinen nicht als unabhängig betrachtet werden können. Viele Entwurfsentscheidungen können als Vorteil für eine Disziplin, jedoch auch als Nachteil für eine andere Disziplin gesehen werden. Betrachtet man beispielsweise eine Linearachse einer Werkzeugmaschine, so kann deren exakte Positionierung rein mechanisch oder mit Hilfe eines Regelkreises, der eine ungenaue Positionierung automatisiert korrigiert, gewährleistet werden. Die korrigierende Variante verringert den mechanischen Aufwand, zwingt jedoch die Elektronik und die Informationstechnik zu Mehraufwand. Es sind also häufig Entscheidungen zu treffen, die nach mehreren, unter Umständen gegensätzlichen Kriterien zu treffen sind. Die Herausforderung hierbei ist, dass die Kriterien auf mehrere Disziplinen verteilt sind und von einer einzelnen Person in der Regel nicht mehr gänzlich erfasst werden können.

Damit kommt man zur zweiten Herausforderung, der Zusammenarbeit der einzelnen Disziplinen im Team. Da eine einzelne Person nicht mehr alle Disziplinen beherrschen kann, wird in Teams entwickelt, in denen die einzelnen Disziplinen durch eine oder mehrere Personen vertreten werden. Grundsätzlich treten hier gruppendynamische Effekte auf und zwischenmenschlichen Herausforderungen muss begegnet werden. Diese Aspekte sind für die vorliegende Arbeit jedoch nicht von Interesse und es sei zum Beispiel auf [Nerd11] oder [SCR08] verwiesen. Der Fokus bezüglich der Arbeit in Teams muss in diesem Zusammenhang eher auf die Tatsache gelegt werden, dass eine Entwicklung nicht von einer einzelnen Person, die alles Fachwissen besitzt, sondern von mehreren Personen, die jeweils anderes Fachwissen besitzen, durchgeführt wird. Auch wenn alle Beteiligten zum selben Team gehören, kann man deshalb schon von einer Art verteilten Entwicklung sprechen. Die einzelnen Teammitglieder müssen sich absprechen und

müssen ihre Ergebnisse und Entscheidungen den anderen Mitgliedern verständlich darlegen. Dies muss während der gemeinsamen Entwicklung geschehen, wobei die einzelnen Disziplinen ähnlich stark gewichtet werden.

Dies führt zur nächsten Herausforderung: Heutzutage verlaufen multidisziplinäre Entwicklungen bezogen auf die einzelnen Disziplinen parallel ab. Das heißt, dass nicht mehr eine einzelne Disziplin zu entwickeln beginnt und die anderen den Entwurf anschließend erweitern, sondern alle Disziplinen gleichzeitig und weitestgehend gleichberechtigt entwickeln. Dieses Vorgehen ermöglicht eine bessere Gesamtlösung, da die Anforderungen bzw. Randbedingungen aller Disziplinen schon frühzeitig berücksichtigt werden können. Jedoch erfordert diese gleichzeitige und gleichberechtigte Entwicklung viele Änderungen des Entwurfs. Die enge Zusammenarbeit zwischen den Disziplinen ist also essentiell.

Dies wird durch eine weitere Herausforderung erschwert, nämlich den unterschiedlichen gebräuchlichen Vorgehensweisen, die in den verschiedenen Disziplinen eingesetzt werden. So werden die Vertreter der einzelnen Disziplinen, in der Regel Ingenieure, unterschiedlich ausgebildet, erlernen selbst innerhalb einer Disziplin unterschiedliche Vorgehensweisen, haben andere Erfahrungen gemacht und besitzen einen unterschiedlichen Wissensstand. Deshalb ist es nicht immer möglich und praktikabel, eine einheitliche Vorgehensweise für die Entwicklung festzulegen. Vielmehr ist es erforderlich, die verschiedenen Vorgehensweisen innerhalb eines gewissen Rahmens aufeinander abzustimmen und vor allem die Konsistenz der Ergebnisse sicherzustellen. Deshalb ist es notwendig, dass alle Beteiligten eine gemeinsame und für alle verständliche Vorstellung des Entwicklungsziels besitzen. Diese Vorstellung muss bereits in den frühen Entwicklungsphasen, in denen alle beteiligten Disziplinen gemeinsam arbeiten und das grobe Konzept erstellen, festgelegt werden. Sie dient als Vision für die in den späteren Entwicklungsphasen in der Regel disziplinspezifisch getrennt verlaufende Entwicklung.

Diese gemeinsame Vorstellung ist jedoch nicht einfach zu realisieren, da in den einzelnen Disziplinen unterschiedliche Begriffswelten existieren. So werden beispielsweise unterschiedliche Begriffe für denselben Sachverhalt benutzt (Synonyme), was oft zu Unklarheiten führt. Weitaus gravierender sind jedoch gleichlautende Begriffe (Homonyme), die für unterschiedliche Sachverhalte benutzt werden. Dies führt unweigerlich zu Missverständnissen und gefährdet jede multidisziplinäre Entwicklung. In vielen Fällen werden solche Situationen erst sehr spät in der Entwicklung oder gar nicht entdeckt. Deshalb ist es wichtig, dass wichtige Begriffe von allen Beteiligten von Anfang an richtig verstanden und gebraucht werden.

Gemeinsam verwendete Begriffe reichen jedoch nicht aus, um eine multidisziplinäre Entwicklung erfolgreich werden zu lassen. Nur wenn jeder seine genaue Aufgabe und die Schnittstellen zu den anderen Disziplinen kennt, kann ein funktionierendes Produkt entstehen. Jede Disziplin hat eine andere Sichtweise auf das spätere Produkt, hat andere Schwerpunkte und andere Modelle, um das Produkt zu beschreiben. Das ist durchaus notwendig, um alle Details zu beschreiben.

Ein einzelnes Gesamtmodell wäre dabei oft überladen und zu komplex. Trotzdem beschreiben alle Modelle dasselbe Produkt und besitzen deshalb Abhängigkeiten. Es wird in diesem Fall von einem System von Modellen gesprochen. Die Abhängigkeiten müssen allen beteiligten Entwicklern klar sein und müssen über die gesamte Projektlaufzeit gepflegt werden. Wenn eine Disziplin in einem Modell etwas ändert, muss diese Änderung von anderen Disziplinen erkannt, verstanden und in den eigenen Modellen nachgepflegt werden. Nur so ist gewährleistet, dass die gemeinsame Vision des Produkts, verteilt auf unterschiedliche Modelle, konsistent ist und zum Abgleich der disziplinspezifischen Entwicklungen herangezogen werden kann.

Eine weitere Herausforderung im Zusammenhang mit der heterogenen Modelllandschaft ist der Einsatz von Werkzeugen für die Modellierung oder das Engineering. Häufig werden für die Erstellung und Pflege der Modelle Werkzeuge unterschiedlicher Hersteller verwendet, die keine oder unzureichend kompatible Schnittstellen besitzen. Ein Abgleich zwischen unterschiedlichen Modellen muss deshalb oft manuell erfolgen, wobei der Faktor Mensch als mögliche Fehlerquelle berücksichtigt werden muss.

## 1.3 Ziel einer rechnergestützten Konsistenzprüfung der Modelle

Mit Hilfe der rechnergestützten Prüfung soll den genannten Herausforderungen begegnet werden und somit multidisziplinäre Entwicklungen, vorgestellt am Beispiel der Mechatronik, unterstützt werden. Der Schwerpunkt liegt dabei auf der rechnergestützten Aufdeckung von Inkonsistenzen zwischen den einzelnen, jedoch zusammenhängenden Entwurfsmodellen, um deren Beseitigung zu ermöglichen und ein konsistentes System der Modelle zu erhalten. Die Arbeit konzentriert sich auf die frühen Phasen des Entwurfs, in denen eine gemeinsame Vorstellung und Referenz für die weiteren Entwurfsphasen erstellt wird (Prinziplösung [GFS05]). Die in Abbildung 1.1 dargestellten Phasen sind [VDI2206] entnommen.

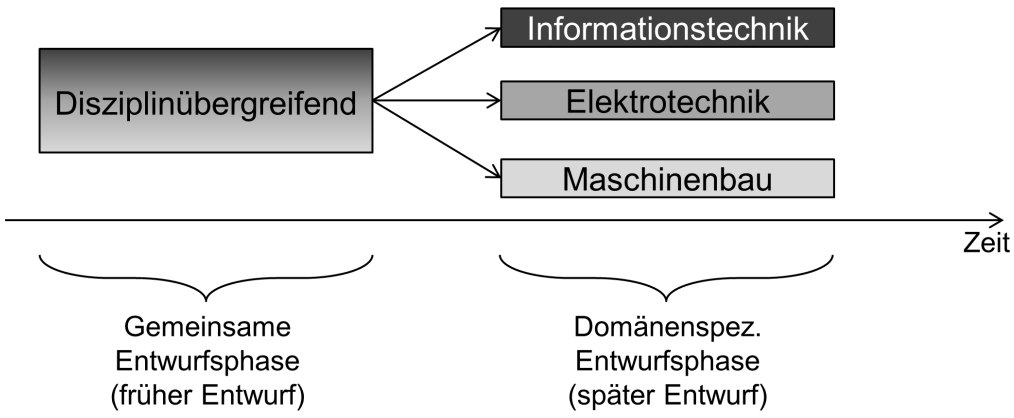


Abbildung 1.1: Entwurfsphasen in der Mechatronik

Ist diese Vorstellung konsistent und wird im Laufe der weiteren Entwicklung, bei der fast zwangsläufig Änderungen an dieser Referenz vorgenommen werden müssen, gepflegt, so wird auch eine disziplinspezifische Prüfung für die nach Disziplinen getrennten weiteren Entwurfs- und Detaillierungsphasen ermöglicht. Diese Prüfung für die späteren Entwurfsphasen ist nicht mehr Teil dieser Arbeit. Ein Beispiel für die Disziplin Maschinenbau wird jedoch in [KRBG11a] beschrieben.

Die Prüfung eines Systems von Modellen auf Konsistenz ist zwar eine zeitaufwendige, jedoch wenig kreative Tätigkeit für einen Entwickler. Die rechnergestützte Prüfung soll die Entwickler deshalb entlasten, indem sie diese Aufgabe übernimmt. Zudem soll sie die Entwickler durch die Bereitstellung von Informationen aus den Modellen unterstützen, aufgetretene Inkonsistenzen zu beseitigen. Des Weiteren soll durch die rechnergestützte Ausführung der Konsistenzprüfung der Einfluss menschlicher Faktoren minimiert werden.

Die Konsistenzprüfung soll zeitlich flexibel im Entwurfsprozess angewendet werden können, auch wenn noch nicht alle Modelle vollständig sind bzw. denselben Detaillierungsgrad besitzen. Das ermöglicht die Aufdeckung von einzelnen Inkonsistenzen bereits während der frühen Entwurfsphase und nicht erst gesammelt an deren Ende.

Damit die rechnergestützte Prüfung im industriellen Umfeld zur Anwendung kommen kann, muss diese mit relativ wenig Aufwand in die bestehenden Entwicklungsprozesse integriert werden können. Bereits die gemeinsame Vorstellung eines Systems besteht aus mehreren Modellen, um alle wichtigen Aspekte ausreichend zu beschreiben. Die Modelle sind meist heterogen, da sie Informationen verschiedener Disziplinen enthalten und oft mit Hilfe unterschiedlicher Engineering Werkzeuge erstellt wurden. Zudem kann es während der Erstellung der gemeinsamen Vorstellung notwendig sein, dem System ein weiteres Modell hinzuzufügen, um einen Aspekt detaillierter zu beschreiben. Die automatisierte Konsistenzprüfung soll also auf ein System von heterogenen Modellen anwendbar sein und flexibel weitere Modelle in die Prüfung aufnehmen können.

## 1.4 Abgrenzung zu ähnlichen Themenstellungen

Das hier vorgestellte Konzept bewegt sich im Gebiet der Konsistenzprüfung und -sicherung von Modellen. Verschiedene andere Ansätze beschäftigen sich ebenfalls mit diesem Themenfeld. Um die Abgrenzung zu erleichtern, werden im Folgenden verwandte Arbeiten vorgestellt:

- *Verknüpfung mittels Modellverbinder*: In [CHF14] wird ein Konzept vorgestellt, das mehrere, voneinander abhängige Modelle mittels eines explizit definierten Modellverbinders verknüpft. Die nachfolgende Arbeit strebt jedoch die flexible Kopplung zwischen den Modellen an und damit die Ermittlung der Verbindungen zwischen Modellen zur Laufzeit.

- *Konsistenz mittels Konsistenzregeln*: Ein Ansatz zur Konsistenzsicherung mit Hilfe modellspezifisch definierter Konsistenzregeln ist in [HEZ10] beschrieben. Es werden dabei jedoch bekannte bzw. gewissen Vorgaben entsprechende Modelle vorausgesetzt.

- *Konsistenz mittels Grammatiken*: Ein anderer Ansatz ist, die Struktur verschiedener Modelle mittels Grammatiken zu vergleichen [GGS+07]. Dabei kommen Triple-Graph-Grammatiken (TGG) zum Einsatz. Dieser Ansatz setzt, im Gegensatz zu dem in der nachfolgenden Arbeit angestrebten Ziel, ebenfalls bekannte Modelltypen voraus.

- *Konsistenzprüfung mit Ontologien*: In [FKV14] wird ein Ansatz zur Konsistenzprüfung von Modellen mittels Ontologien vorgestellt. Dazu werden die in SysML4Mechatronics (Erweiterung von SysML) formulierten Modelle in Ontologien abgebildet und mittels Abfragen an die Ontologie Inkonsistenzen aufgedeckt. Jede Abfrage sucht dabei eine oder mehrere Inkompatibilitäten in den Modellen. Der beschriebene Stand erfordert jedoch noch das explizite Hinterlegen der Ontologieabfragen (Konsistenzregeln) und das Vorliegen der Modelle in SysML4Mechatronics. Heterogene Modelle werden somit noch nicht unterstützt.

## 1.5 Gliederung der Arbeit

Die vorliegende Arbeit beschreibt in 11 Kapiteln, wie die gewünschten Ziele unter der Berücksichtigung der beschriebenen Herausforderungen erreicht werden können. Dazu werden im zweiten Kapitel die Grundlagen der Mechatronik vorgestellt. Neben der Erklärung, was die Mechatronik ist und wie sie entstanden ist, wird das Vorgehen bei der Entwicklung mechatronischer Produkte oder Systeme erläutert und mehrere Modellierungsansätze für die frühe Entwurfsphase der Mechatronik werden präsentiert.

Das dritte Kapitel beschreibt die Grundlagen zu den im Konzept eingesetzten Softwareagenten. Nach einer Einführung in die agentenorientierte Softwareentwicklung werden die Konzepte, die einen Agenten ausmachen, aufgezeigt. Im Anschluss daran folgen der Aufbau eines einzelnen Agenten sowie der eines Agentensystems. Schließlich werden einige Beispiele des Einsatzes von Agenten in der Automatisierungstechnik bzw. zur Automatisierung von zeitaufwändigen Tätigkeiten präsentiert.

Das in dieser Arbeit beschriebene Konzept basiert auf der Nutzung von Ontologien, die im vierten Kapitel vorgestellt werden. Nach einer Einführung in die Grundidee von Ontologien wird deren Aufbau beschrieben. Darauf folgt die Beschreibung des Vorgehens zur Erstellung von Ontologien. Abgeschlossen wird das Kapitel durch einige Anwendungsbeispiele.

Im fünften Kapitel werden bestehende Konzepte zur Konsistenzsicherung von Modellen erläutert. Dazu werden zunächst die möglichen Inkonsistenzen klassifiziert. Im Anschluss daran werden Konzepte von der manuellen über die halbautomatisierte bis hin zur vollautomatisierten

Konsistenzsicherung vorgestellt. Schließlich werden die Anforderungen an diese Arbeit abgeleitet.

Das sechste Kapitel bietet eine Übersicht zur automatisierten Prüfung von Modellen. Im Vergleich mit dem menschlichen Vorgehen wird das rechnergestützte Konzept präsentiert. Anschließend wird auf die speziellen Voraussetzungen beim Umgang mit heterogenen Modellen hingewiesen. Schließlich werden die notwendigen Vorarbeiten zur Formalisierung der Entwurfsmodelle vorgestellt, ohne die eine rechnergestützte Prüfung der Modelle nicht möglich ist.

Der erste Teil des Konzepts, der das die Prüfung ausführende Agentensystem beschreibt, befindet sich im siebten Kapitel. Zunächst wird ein Überblick über das System geschaffen. Danach werden die Aufgaben bzw. Rollen der Agenten abgeleitet und die verschiedenen Agententypen vorgestellt. Die Festlegung der Interaktionen zwischen den Agenten schließt das Kapitel ab.

Das achte Kapitel beschreibt den zweiten Teil des Konzepts und stellt vor, wie das für eine Prüfung benötigte Wissen mit Hilfe von Ontologien abgebildet wird. Nach einer Übersicht werden zunächst die Beschreibung von Modellen und der zugehörige Aufbau einer lokalen Ontologie erläutert. Im Anschluss daran folgen die Abbildung von Regeln, deren Einhaltung bei einer Prüfung ermittelt wird, und der Aufbau der globalen Ontologie.

Im Rahmen dieser Arbeit wurde das erstellte Konzept prototypisch umgesetzt. Der entsprechende Prototyp wird im neunten Kapitel präsentiert. Zunächst wird der Editor zur Erstellung und Bearbeitung der mechatronischen Modelle vorgestellt. Die Beschreibung der Implementierung des Agentensystems folgt und das Kapitel wird durch die Präsentation der Realisierung der Ontologien abgeschlossen.

Im zehnten Kapitel wird das Konzept mit Hilfe eines Anwendungsbeispiels bewertet. Zuerst wird das Beispielszenario erläutert. Im Anschluss daran folgt der Ablauf der Prüfung. Schließlich werden die Ergebnisse der beispielhaften Anwendung und die dabei gemachten Erfahrungen beschrieben.

Das elfte und letzte Kapitel enthält eine Zusammenfassung der wesentlichen Aspekte der vorliegenden Arbeit. Das Konzept wird abschließend bewertet und die notwendigen Voraussetzungen sowie die Grenzen werden aufgezeigt. Ein Ausblick auf mögliche zukünftige Forschungsarbeiten zur Weiterentwicklung und Übertragung des Konzepts schließen das Kapitel ab.

# 2 Grundlagen zur Mechatronik

Die Mechatronik wird in der vorliegenden Arbeit als Beispiel für eine multidisziplinäre Entwicklung im Umfeld der Automatisierungstechnik genutzt. Deshalb wird im Folgenden beschrieben, was die Mechatronik ausmacht und wie sie entstanden ist. Das Vorgehen und die Besonderheiten bei der Entwicklung von mechatronischen Systemen werden ausgeführt und Begrifflichkeiten geklärt. Vor allem der Einfluss der Multidisziplinarität auf die Entwicklung und wie die Entwickler damit umgehen, ist für den weiteren Verlauf der Arbeit von Bedeutung. Sie hat beispielsweise zur Entstehung verschiedener Entwurfsmodelle beigetragen, die den Entwicklern mehr oder weniger Freiheiten bei der Modellierung lassen und damit zu einer Heterogenität in der Modelllandschaft führen. Mehrere Beispiele solcher Entwurfsmodelle werden dazu vorgestellt. Zunächst folgt jedoch eine Einführung in die Mechatronik.

## 2.1 Einführung in die Mechatronik

Unter dem Begriff Mechatronik versteht man eine Kombination mehrerer ingenieurswissenschaftlicher Disziplinen und deren nutzbringende Zusammenarbeit. Viele heutige Produkte können als mechatronische Systeme identifiziert werden, oft auch das Produktionssystem, mit dem sie gefertigt werden [Bish07].

Der Japaner Ko Kikuchi prägte den Begriff „Mechatronics“ 1969 [Come94]. Als Hersteller von Servoantrieben und Robotern bezeichnete er damit die Erweiterung mechanischer Komponenten um elektronische Funktionen. Der Begriff setzte sich damals aus „mechanism“ und „electronics“ zusammen und war zeitweise als Handelsname geschützt.

Dabei war die Mechanik die führende Disziplin und die Elektronik wurde lediglich auf eine bestehende, fixe mechanische Konstruktion aufgesetzt. So konnten zwar gewisse Synergieeffekte genutzt werden, eine echte Verschmelzung der Disziplinen wurde jedoch noch nicht erreicht.

Mit der rasanten Entwicklung der Mikroelektronik, die immer leistungsstärkere Mikrocontroller ermöglichte, ist auch die Informationstechnik mehr und mehr in die Mechatronik integriert worden. Der Begriff Mechatronik setzt sich heute also aus der Mechanik, der Elektrotechnik und der Informationstechnik zusammen [VDI2206] und bezeichnet die Zusammenarbeit dieser Disziplinen. Ende der 90iger Jahre wurde in [Iser99] folgende Abbildung zur Darstellung der Mechatronik benutzt:

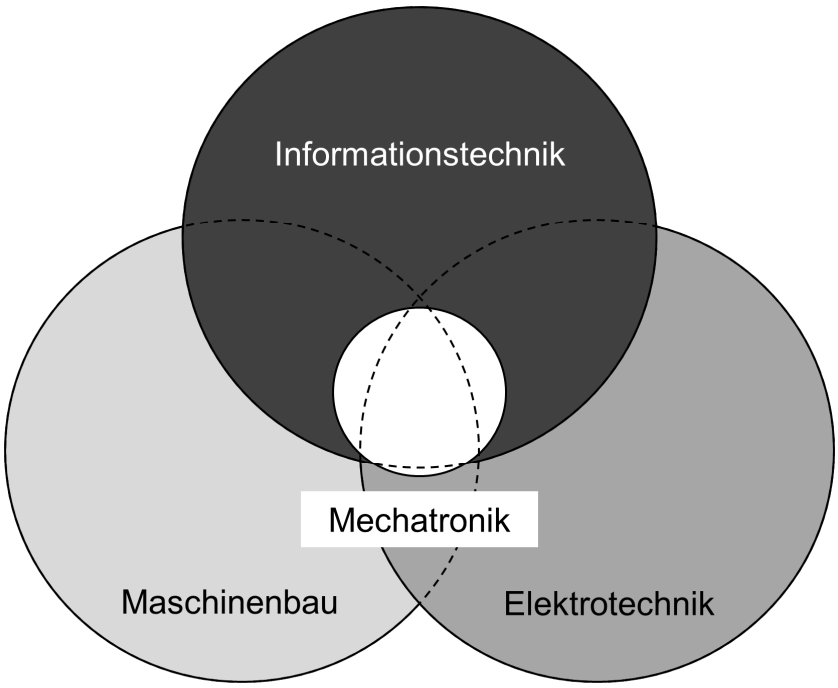


Abbildung 2.1: Mechatronik nach [Iser99]

Obwohl alle drei Disziplinen mit Kreisen derselben Größe dargestellt wurden, blieb die Mechanik lange Zeit die führende Disziplin, auf die die anderen Disziplinen aufbauen mussten. So wurde in der Regel sequentiell entwickelt, indem die Mechanik mit der Entwicklung begonnen hat, die Elektrotechnik später auf die Ergebnisse aufsetzte und am Ende die Informationstechnik ergänzt wurde. In Abbildung 2.2 wird dieser Ablauf graphisch dargestellt. Da die Elektrotechnik und die Informationstechnik jeweils auf bestehende Entwicklungsstufen aufsetzen mussten, konnten zwar gewisse Synergieeffekte genutzt werden, optimale Lösungen wurden jedoch nicht erzielt [VDI2206].

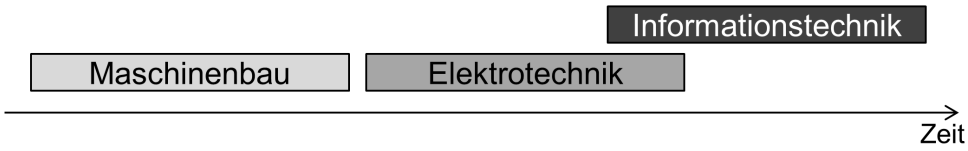


Abbildung 2.2: Sequentielles Engineering in der Mechatronik

Deshalb wird heutzutage eine gemeinsame, gleichzeitige und gleichberechtigte Entwicklung aller Disziplinen angestrebt. Van Brussel spricht als einer der ersten vom „Concurrent Engineering“ in diesem Bereich [Brus96] (vgl. Abbildung 2.3). Somit kann jede Disziplin Einfluss auf die Entwicklungsergebnisse der anderen Disziplinen nehmen und die jeweiligen Technologien können besser und effizienter genutzt werden. Eine Integration der zuvor mehr oder weniger unabhängigen Disziplinen und ihrer Technologien wird so ermöglicht. Dies führt letztendlich zu optimierten Produkten, erhöht jedoch gleichzeitig die Komplexität bei der Entwicklung.

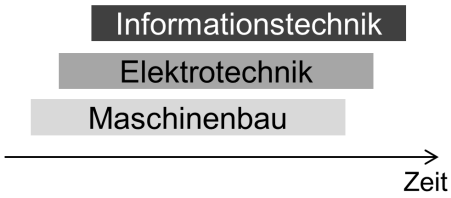


Abbildung 2.3: Concurrent Engineering in der Mechatronik

Mit den Fortschritten der einzelnen Disziplinen bezüglich neuer Technologien etc. hat sich auch die Mechatronik weiterentwickelt [Moeh09]. So wurden weitere Rand- oder Teildisziplinen integriert, die sich im Laufe der Zeit zu eigenen Disziplinen entwickelt haben. Beispiele sind die Mikromechanik, Fluidmechanik oder Elektromechanik. Zudem werden je nach Entwicklungsprojekt auch zunächst eher fremde Disziplinen berücksichtigt, wie Biotechnologien oder die Flugmechanik etc.

Die Grenzen der Mechatronik sind fließend. Bis heute hat sich keine einheitliche Definition herausgebildet. Neuere Ansätze für eine Definition haben jedoch den Trend gemeinsam, dass die Mechatronik mehr als die Summe ihrer Einzelteile (Disziplinen) ist. Sie ist zu einer neuen Denkweise geworden, die heutzutage treffender mit folgender Abbildung dargestellt wird:

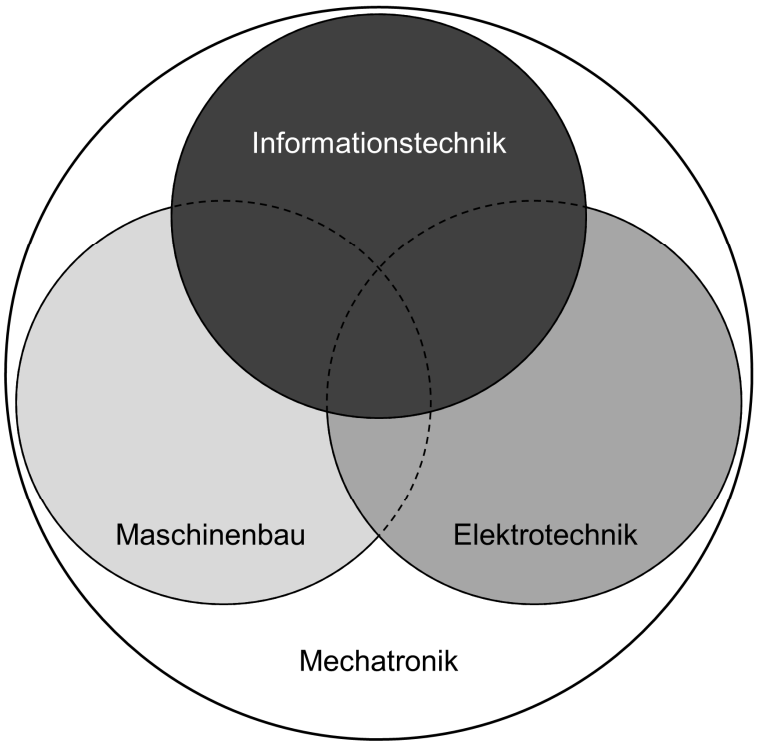


Abbildung 2.4: Mechatronik heute (nach [Clus14])

Die grundlegende Struktur mechatronischer Systeme wird in der VDI-Richtlinie 2206 [VDI2206] gemäß Abbildung 2.5 dargestellt. Dabei besteht ein mechatronisches System aus einem Grundsystem, welches in der Regel eine mechanische Struktur besitzt. Es kann sich jedoch auch beispielsweise um ein elektromechanisches oder pneumatisches System oder ein Sys-

tem aus anderen Kombinationen handeln. Allgemein kann hier also ein beliebiges physikalisches System als Grundsystem angenommen werden.

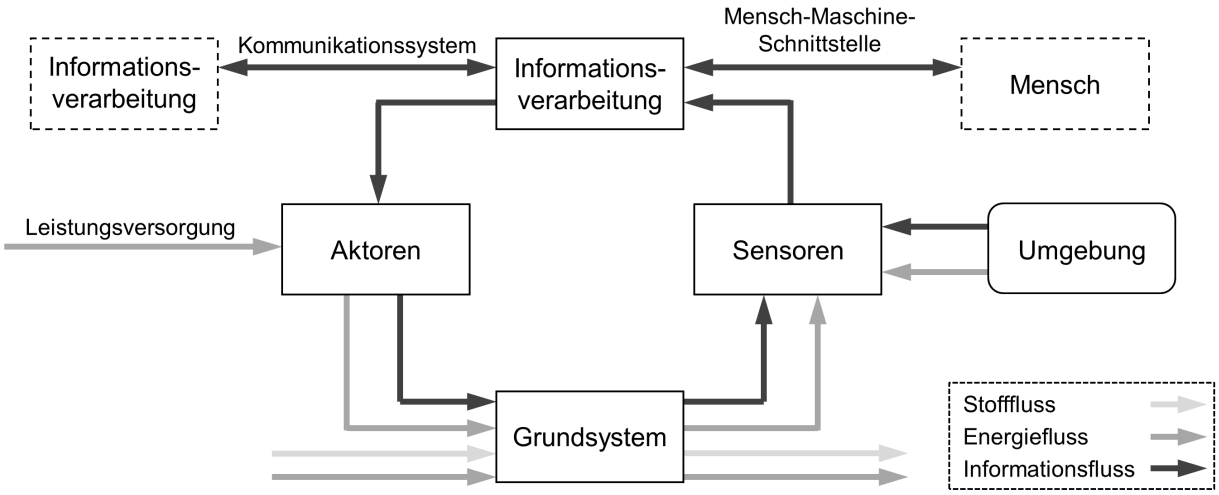


Abbildung 2.5: Grundstruktur eines mechatronischen Systems [VDI2206]

Sensoren erfassen die notwendigen Zustandsgrößen des Grundsystems und der Umgebung. Sie können beispielsweise physikalische Größen messtechnisch erfassen, aber auch virtuelle, das heißt errechnete Größen, wiedergeben. Die Informationen der Sensoren werden von der Informationsverarbeitung benötigt. Diese befindet sich heutzutage in der Regel auf einem Mikrocontroller und arbeitet damit digital, kann aber auch analog ausgeführt sein. Je nach System existiert zur Bedienung eine Mensch-Maschine-Schnittstelle, die die Verbindung zwischen Mensch und Informationsverarbeitung herstellt. Zudem kann die Informationsverarbeitung des (lokalen) Systems mit Hilfe eines Kommunikationssystems mit weiteren Systemen verbunden sein. Diese Betrachtung ermöglicht es, Systeme hierarchisch in Teilsysteme zu zerlegen oder Systeme zu vernetzen. Die Informationsverarbeitung bestimmt an Hand der ihr vorliegenden Informationen die notwendigen Eingriffe in das Grundsystem. Aktoren führen diese Eingriffe aus und können das Grundsystem aktiv beeinflussen. Sie schließen somit den Regelkreis. Die benötigte Energie beziehen sie von einer (bezüglich des Grundsystems) externen Energieversorgung.

Die hier dargestellten klaren Grenzen zwischen den einzelnen Elementen eines mechatronischen Systems verschwimmen heutzutage, wenn man sogenannte intelligente Sensoren oder Aktoren betrachtet. Sie haben, wenn man sie als eine Einheit betrachtet, bereits eine Informationsverarbeitung in Form eines Mikrocontrollers integriert, der beispielsweise die Anbindung an ein Bussystem realisiert. Je nach Abstraktionsgrad sind diese Elemente also nicht immer eindeutig zu trennen.

Der Austausch zwischen den Elementen findet über Stoff-, Energie- und Informationsflüsse statt. Dabei sind Stoffflüsse jegliche Form von Material, egal ob fest, flüssig oder gasförmig, das von einem Element zum nächsten transportiert wird. Mit dem Energiefluss wird zum Beispiel die mechanische, thermische oder elektrische Energie bezeichnet, die zwischen den Elementen

übertragen wird. Je nach Detaillierungsgrad kann hiermit auch direkt eine Kraft oder ein elektrischer Strom gekennzeichnet sein. Der Informationsfluss beinhaltet alle Informationen (Messgrößen, Stellgrößen oder allgemein Daten), die zwischen den Elementen ausgetauscht werden.

## 2.2 Vorgehen beim mechatronischen Entwurf

Um ein disziplinübergreifendes System zu entwickeln, ist es notwendig, dass eine einheitliche und vorher abgestimmte Entwurfsmethodik angewendet wird. Auf Grund der in ihrer jeweiligen Ausprägung sehr unterschiedlichen mechatronischen Systeme (von Waschmaschinen über Automobile bis hin zu komplexen Fertigungs- oder Raffinierungsanlagen), muss diese Entwurfsmethodik Freiheitsgrade für die Anpassung an den jeweiligen Anwendungsfall und eine entsprechende Skalierbarkeit aufweisen, um sinnvoll eingesetzt werden zu können. Der Verein Deutscher Ingenieure (VDI) schlägt in der [VDI2206] ein Vorgehen basierend auf dem V-Modell, allgemeine Problemlösungszyklen und vordefinierte Prozessbausteine vor. Diese Elemente werden im Folgenden vorgestellt.

### 2.2.1 V-Modell

Das V-Modell stammt ursprünglich aus der Softwareentwicklung und wurde dort eingeführt, um die Durchführung von Softwareprojekten zu verbessern. In der Mechatronik wird das V-Modell in adaptierter Form für den übergeordneten Entwurfsprozess (Makrozyklus) eingesetzt.

Das V-Modell hat seinen Namen durch seine V-Struktur (vgl. Abbildung 2.6) erhalten, bei der zunächst top-down (vom Groben ins Feine) entworfen und anschließend bottom-up (vom Feinen ins Grobe) verifiziert wird. Basis der Entwicklung sind die zugrunde liegenden Anforderungen. Diese Anforderungen beschreiben beispielsweise wie das Produkt auszusehen hat und welche Funktionen es besitzen muss. Sie werden zudem am Ende des Makrozyklus herangezogen, um das entstandene Produkt zu bewerten.

Im ersten Entwurfsschritt, dem Systementwurf, wird ein domänenübergreifendes Lösungskonzept, die Prinziplösung, erstellt. Dieses enthält die wesentlichen physikalischen und logischen Wirkungsweisen sowie die einzusetzenden Lösungsprinzipien. Diese frühe Entwurfsphase wird domänenübergreifend durchgeführt, das heißt Ingenieure aller notwendiger Disziplinen sind daran beteiligt.

Im nächsten Entwurfsschritt, dem domänenspezifischen Entwurf, wird das Lösungskonzept detailliert. In dieser Phase arbeiten die Ingenieure der einzelnen Disziplinen in der Regel weitgehend getrennt voneinander und benutzen disziplinspezifische Modelle, da mit dem Lösungskonzept eine gemeinsame Referenz besteht, an der sie sich orientieren können. Dies ermöglicht nicht nur die nach Domänen getrennte Entwicklung, sondern auch die Zerlegung des mechatronischen Systems in Module, die wiederum getrennt voneinander entwickelt werden können.

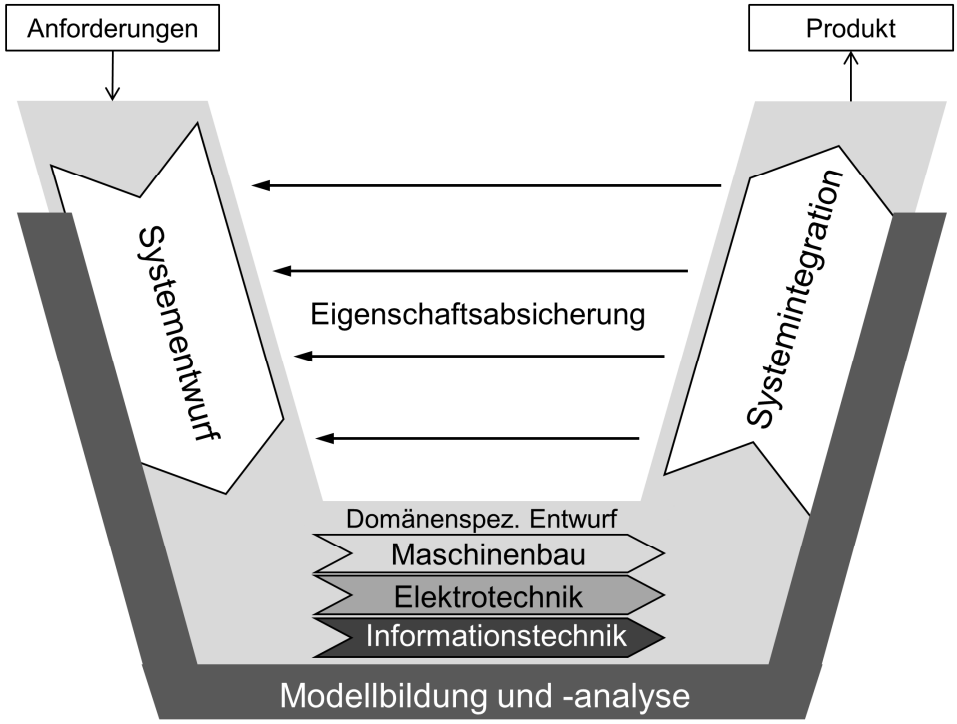


Abbildung 2.6: V-Modell nach [VDI2206]

Im dritten Entwurfsschritt, der Systemintegration, werden die einzelnen Ergebnisse zum Gesamtsystem zusammengeführt. Dieser Prozess muss nicht zwangsweise in einem Schritt durchgeführt werden, sondern es ist möglich und sinnvoll, zunächst einzelne Module oder Modulgruppen zu bilden und diese nach und nach zum Gesamtsystem zusammenzuführen.

Während des gesamten Entwurfsprozesses muss sichergestellt werden, dass die jeweiligen Entwurfsergebnisse den Anforderungen und dem erstellten Lösungskonzept entsprechen. Sollte dies nicht der Fall sein, so muss der Entwurf korrigiert werden oder, sofern das z. B. aus technischen, physikalischen oder finanziellen Gründen nicht möglich ist, über eine Änderung des Lösungsprinzips oder gar der Anforderungen nachgedacht werden. Diese permanente Eigenschaftsabsicherung hilft jedoch, den Entwurfsfortschritt permanent zu überwachen und die Ziele im Auge zu behalten.

Am Ende des Entwurfsprozesses steht das Produkt, das mechatronische System. Dieses ist in der Regel jedoch noch nicht serienreif, sondern muss durch einen oder mehrere weitere Makrozyklen (weitere Durchläufe des Entwicklungsprozesses) verbessert und bis zur Produktreife gebracht werden. Das V-Modell wird also vor allem bei komplexeren mechatronischen Systemen iterativ mehrfach durchlaufen, bis am Ende das gewünschte Produkt entstanden ist.

Da eine Entwicklung ohne Dokumentation und zumindest theoretischer Versuche kaum möglich ist, wird der Entwurfsprozess durch die Modellbildung und -analyse begleitet [Jans10]. In verschiedenen Modellen werden die Entwurfsergebnisse für die weitere Verwendung dokumentiert. Diese können beispielsweise als Referenz für spätere Entwicklungsphasen oder zu Simulationszwecken herangezogen werden, um Eigenschaften simulativ zu verifizieren [AABL13].

### 2.2.2 Problemlösungszyklus

Im Vergleich zum V-Modell als Makrozyklus ist ein Problemlösungszyklus ein Mikrozyklus. Er unterstützt die Entwickler beim Bearbeiten geplanter Teilaufgaben, jedoch auch bei unvorhergesehenen Problemen. Der in der [VDI2206] vorgestellte Problemlösungszyklus (siehe Abbildung 2.7) stammt aus dem Systems Engineering und wird in ähnlicher Ausführung auch in anderen Bereichen eingesetzt.

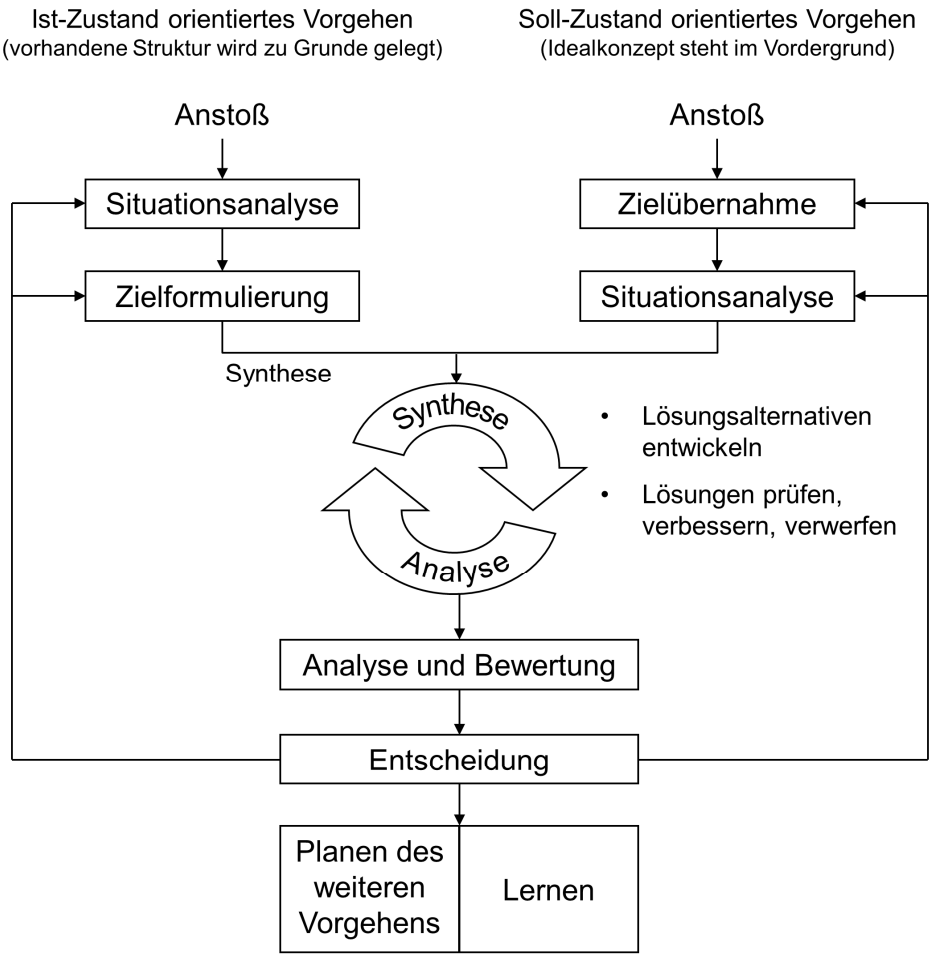


Abbildung 2.7: Problemlösungszyklus nach [VDI2206]

Der Zyklus beginnt entweder mit der Situationsanalyse oder mit der Zielübernahme. Die Situationsanalyse wird gewählt, wenn die Situation unklar ist und das Ziel ermittelt werden muss (Ist-Zustand orientiertes Vorgehen). Ist das Ziel vorgegeben (Soll-Zustand orientiertes Vorgehen), so wird mit der Zielübernahme gestartet und die Situation anschließend analysiert.

Im Anschluss daran werden Lösungen für die gegebene Aufgabenstellung gesucht. Dazu werden abwechselnd Lösungsvarianten ermittelt und deren Anwendbarkeit analysiert (Synthese und Analyse). Dieses Suchen und Bewerten wird von den Entwicklern zum großen Teil bewusst, aber auch unbewusst durchgeführt. Ziel des iterativen Problemlösungsschrittes ist es, verschiedene Lösungsvarianten zu erarbeiten, aus denen am Ende die am besten passende Lösung ausgewählt werden kann. Bei der Lösungssuche können neue Aspekte als relevant erkannt werden und das Ziel oder die Bewertungskriterien müssen dementsprechend angepasst werden.

Wurde eine ausreichende Anzahl an Lösungsvarianten ermittelt, werden diese an Hand der gestellten Anforderungen detailliert analysiert und bewertet. Diese Analyse kann mit Hilfe von theoretischen Berechnungen, Simulationen oder Versuchen erfolgen. Unter Umständen ist ein Rücksprung zur Lösungssuche erforderlich, sofern sich der Konkretisierungsgrad der Lösungen zu sehr unterscheidet. Am Ende dieses Schrittes ist die bevorzugte Lösungsalternative (evtl. auch mehrere) ermittelt. Nun wird entschieden, ob mit dieser Lösungsvariante in der Entwicklung fortgefahren werden kann oder ob die Situationsanalyse und Zielformulierung erneut durchlaufen werden müssen.

Abgeschlossen wird der Mikrozyklus durch das Planen des weiteren Vorgehens und das Lernen. Die Planung wird in der Regel die Lösung von weiteren Problemstellungen (weitere Mikrozyklen) vorsehen und somit einen effizienten und angepassten Entwurfsprozess ermöglichen. Unter dem Begriff Lernen verbirgt sich der kritische Rückblick auf den Prozess und dessen Ergebnisse. So lässt sich Wissen für zukünftige Problemlösungsprozesse generieren und diese können systematisch verbessert werden.

### 2.2.3 Prozessbaustein

Prozessbausteine bezeichnen eine definierte Vorgehensweise für wiederkehrende Arbeitsschritte. Sie beschreiben einzelne Teilschritte sehr viel konkreter als sie im V-Model oder im Mikrozyklus definiert werden können. In Abbildung 2.8 wird beispielhaft der Prozessbaustein Systementwurf vorgestellt.

Beim Systementwurf werden zunächst die in der Anforderungsliste beschriebenen Vorstellungen abstrahiert, um den möglichen Lösungsraum nicht unnötigerweise einzuschränken. Ziel ist es, die Anforderungen auf die wesentlichen Aussagen zu reduzieren und das Problem lösungsneutral zu beschreiben. Dieses Vorgehen ist unter anderem in [FeGr13] detailliert beschrieben. Anschließend wird die Funktionsstruktur aufgestellt. Dazu wird die Gesamtfunktion aus der Problemstellung abgeleitet und in Teilfunktionen zerlegt. Mit Hilfe von Stoff-, Energie- und Informationsflüssen werden die einzelnen Teilfunktionen zur Funktionsstruktur verknüpft. Die Funktionsstruktur wird soweit detailliert, bis es möglich ist, Wirkprinzipien und Lösungselemente zur Realisierung der Teilfunktionen zu ermitteln. Hierbei kommen die Problemlösungszyklen aus dem vorherigen Abschnitt zur Anwendung. Bei der Suche nach Wirkprinzipien und

Lösungselementen ist zu beachten, dass nicht davon ausgegangen werden kann, jede Teilfunktion durch ein einzelnes Lösungselement erfüllen zu können. Dagegen kann es vorkommen, dass einige Lösungselemente mehrere Teilfunktionen erfüllen.

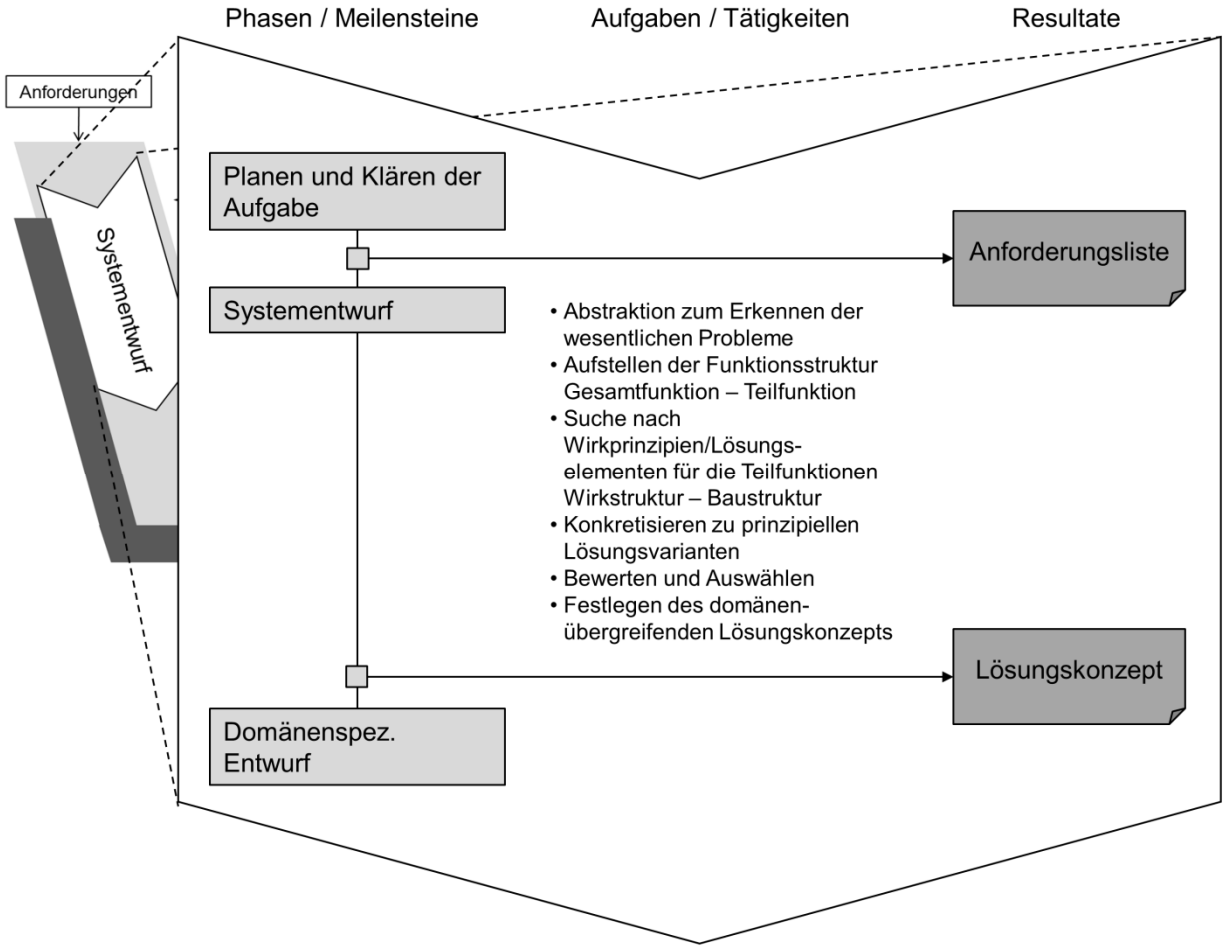


Abbildung 2.8: Prozessbaustein Systementwurf nach [VDI2206]

Sind für alle Teilfunktionen Lösungselemente gefunden, werden diese zur Wirkstruktur verknüpft. Durch überschlägige Rechnungen werden die Lösungselemente dimensioniert. Schließlich wird die Baustruktur (geometrische Anordnung) definiert, um die groben geometrischen Abmessungen und die Lage der einzelnen Lösungselemente zueinander zu erhalten. Da in der Regel so noch kein domänenübergreifendes Konzept festgelegt werden kann, werden die erarbeiteten Modelle weiter detailliert. Dabei werden Aspekte wie Störanfälligkeit, Gewicht oder Lebensdauer berücksichtigt. Ergebnis des Systementwurfs ist das Lösungskonzept oder die Prinziplösung. Sie beschreibt die wesentlichen physikalischen und logischen Wirkungsweisen sowie die Art und Anordnung der Komponenten des zu entwickelnden mechatronischen Systems.

## 2.3 Mechatronische Entwurfsmodelle

Um ein mechatronisches System während des Entwurfs, aber auch später im Betrieb zu beschreiben, sind Modelle notwendig. Da die Beschreibungsmodelle der einzelnen Disziplinen dazu nicht ohne weiteres ausreichen, sind im Laufe der Zeit verschiedene spezifische Konzepte entstanden, mechatronische Systeme zu beschreiben. In den folgenden Abschnitten werden einige dieser Konzepte vorgestellt.

Um diese Konzepte in den richtigen Kontext zu setzen, muss zuvor der Unterschied zwischen dem Inhalt und der Beschreibung oder Notation eines Modells hervorgehoben werden. Der Inhalt bestimmt, was im Modell dargestellt wird und legt damit fest, welche Aspekte oder Sichtweisen auf das mechatronische System beschrieben werden. Ein mechatronisches System wird in der Regel durch mehrere Modelle, die unterschiedliche Aspekte beschreiben, dargestellt, da die Darstellung sonst entweder nicht ausreichend oder zu komplex ist. Die Beschreibung eines Modells legt in diesem Zusammenhang dagegen fest, wie dessen Inhalt beschrieben wird. So kann ein Modell beispielsweise eine graphische Darstellung besitzen oder aus einer rein textuellen Beschreibung bestehen. Die meisten Modelle bestehen, der Übersichtlichkeit und der Verständlichkeit wegen, aus einer graphischen Notation, die durch einen Editor in eine textuelle Beschreibung wie beispielsweise SysML [OMG12] übersetzt wird. Da mit Hilfe von heutigen Computern der Komplexität und dem Umfang der Beschreibung eigentlich keine Grenzen mehr gesetzt sind, ist dieser Aspekt nicht im Fokus dieser Arbeit.

Der Inhalt eines Modells ist dagegen entscheidend, wenn es darum geht, geeignete Modelle zur Beschreibung eines Sachverhalts auszuwählen. Zur Beschreibung von mechatronischen Systemen existieren einige Ansätze, die unterschiedliche Modelle in unterschiedlichen Ausprägungen beinhalten. Im Folgenden werden mehrere dieser Ansätze vorgestellt, um die Unterschiede in der Modellierung darzustellen. Dabei ist festzustellen, dass ein einzelnes, dafür aber allübergreifendes Modell zur Darstellung von mechatronischen Systemen nur für sehr einfache Systeme möglich ist, da die Komplexität schnell ansteigt. In der Regel ist es nicht möglich, alle für die Entwicklung notwendigen Aspekte in einem einzelnen Modell darzustellen. So bestehen alle vorgestellten Ansätze aus mehreren Modellen, besitzen jedoch unterschiedliche Schwerpunkte.

### 2.3.1 Funktionsbeschreibung

Einige Ansätze zur Modellierung von mechanischen und mechatronischen Systemen legen den Fokus auf die Beschreibung der Funktion des Systems. Auch wenn daraus weitere Modelle abgeleitet werden, bleibt die Basis die zu anfangs definierte Funktionsstruktur.

In der VDI-Richtlinie 2221 „Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte“ wird ein Funktionsmodell („Funktionsstrukturen“) erstellt, in dem die Funktionen und deren Strukturen definiert werden [VDI2221]. Ausgehend von der Aufgabenstellung bzw.

der Anforderungsliste, wird untersucht, welche Funktionen im späteren System benötigt werden und zunächst die Gesamtfunktion ermittelt. Diese wird anschließend in Teilfunktionen zerlegt. Der so entstandene hierarchische Funktionsbaum kann über mehrere Stufen bis hin zu Elementarfunktionen erweitert werden. Funktionen werden in [VDI2221] wie folgt definiert: „Lösungsneutral beschriebene Beziehungen zwischen Eingangs-, Ausgangs- und Zustandsgrößen eines Systems“. Eine Funktion wird über ein Verb (z. B. Kraft „leiten“) bzw. über ein substantiviertes Verb (z. B. „Leiten“) definiert. Koller und Kastrup stellen die Elementarfunktionen zum Beispiel in [KoKa94, Koll98] vor. Diese wurden in [HSM+02] bezüglich dem technischen Fortschritt aktualisiert bzw. erweitert.

Im vorgestellten Schritt werden die Funktionen nicht nur ermittelt, sondern über die Baumstruktur hinaus auch in Zusammenhang gebracht. So wird die Struktur der Funktionen, also beispielsweise Abhängigkeiten zwischen den einzelnen Funktionen, definiert.

Für die Funktionen werden anschließend prinzipielle Lösungen gesucht. Das bedeutet, dass zunächst physikalische, chemische oder andere Effekte ausgewählt werden, die einzeln oder in Kombination die gewünschte Funktion erbringen können. Daraus wird eine Wirkstruktur erstellt, die beschreibt, wie die einzelnen Effekte gemäß der Funktionsstruktur zusammenwirken. Die Funktionsstruktur beschreibt dabei, welche Funktionen realisiert werden sollen. Die Wirkstruktur wird genutzt, um festzulegen, wie die Funktionen realisiert werden. Diese kann laut [VDI2221] als Prinzipskizze, Schaltung oder textuelle Beschreibung dokumentiert sein.

In einem weiteren Schritt wird die Wirkstruktur schließlich in einzelne Module gegliedert, die später weitgehend getrennt realisiert werden können. Somit ist es möglich, die Komplexität des Gesamtsystems auf mehrere Teilsysteme oder Module zu verteilen und so beherrschbar zu machen. Die [VDI2221] schlägt als Beschreibungsmittel unter anderem Anordnungsskizzen, Graphen, Logikpläne, Struktogramme oder Fließbilder vor.

[ZLS05] ergänzen zur Funktions- und Strukturbeschreibung die Beschreibung des Zustands und des Verhaltens für einzelne Elemente im mechatronischen System. Sie gehen davon aus, dass jedes Element einen inneren Zustand besitzt. Dieser kann für einen Kondensator beispielsweise die auf ihm gespeicherte Ladung sein, für ein Gelenk dessen derzeitige Position. Diese inneren Zustandsgrößen können wiederum Eingangsgrößen für das Verhalten des jeweiligen Elements sein. Das Verhalten beschreibt die Reaktion des Elements auf äußere Stimuli und somit die Veränderung seines inneren Zustands. Da dieser in der Regel nach außen sichtbar ist (z. B. Position eines Gelenks), kann auch lediglich vom Zustand gesprochen werden.

In einem Modellierungsansatz von [ZTBD02] wird zusätzlich die Umgebung des Systems beschrieben. Diese stellt zum Beispiel elektrische Energie oder Druckluft bereit. Sie kann jedoch auch mehr oder weniger komplexe Funktionen bereitstellen, die beispielsweise durch eine menschliche Bedienperson durchgeführt werden.

[NSH+08] stellen ein Function Design Framework vor, das ebenfalls auf der Funktionsbeschreibung basiert. Sie modellieren ähnliche Aspekte wie [ZTBD02], unterscheiden bei der Modellierung jedoch zwischen Funktionen und Prozessen. Funktionen sind vom System ausgeführte, gewünschte Tätigkeiten (z. B. „Scheibe reinigen“), Prozesse treten in der Umgebung auf (z. B. „Scheibe verschmutzen“). Dazu werden explizite Grenzen für das System, für den (z. B. Produktions-)Prozess und für die unmittelbare Umgebung des Systems eingeführt, um diese Bereiche klar abtrennen zu können.

Keiner der vorgestellten Ansätze schlägt eine konkrete Beschreibungsform oder Notation vor. Sie beschränken sich lediglich auf grobe Andeutungen, wie ein solches Modell aufgebaut sein könnte, und darauf den zu beschreibenden Inhalt zu benennen. Wie die entsprechenden Modelle aussehen, bleibt also weitgehend dem Entwickler überlassen.

### 2.3.2 A3 Architektur-Übersichten

Ursprünglich aus dem Bereich der Dokumentation von Prozessen und Problemlösungen stammen die in [BoBo10] vorgestellten A3 Architektur-Übersichten (A3 Architecture Overviews). Hierbei wird jeweils ein Architektur-Aspekt auf einem Blatt im DIN A3 Format dargestellt. So ist sichergestellt, dass nicht zu viele Informationen auf einmal zu erfassen sind und die Komplexität beherrschbar bleibt. Für größere Systeme sind zur vollständigen Dokumentation jedoch einige dieser Übersichten notwendig, die hierarchisch strukturiert werden können.

Eine A3 Architektur-Übersicht besteht aus einem Blatt Papier (DIN A3 Format), dessen Vorder- und Rückseite benutzt wird. Die Vorderseite enthält die textuelle Beschreibung des Architektur-Aspekts. Hierbei sind folgende Abschnitte vorgesehen:

- Definitionen und Abkürzungen
- Einleitung
- Systemverteilung
- Funktionale Sicht (optional)
- Physikalische Sicht (optional)
- Systeminteressen
- Schlüsselparameter und Anforderungen
- Versionsverwaltung und Autorinformationen
- Entwurfsstrategien, Vorschläge und bekannte Probleme
- Roadmap
- Referenzen

Auch wenn die Vorderseite die textuelle Beschreibung enthält, ist ausdrücklich erwünscht, dass auch graphische Elemente enthalten sind. Der Autor weist dabei auf den bei gleichem Platzbedarf dichteren und damit höheren Informationsgehalt einer Graphik hin [Borc11]. Zudem ist der

Umfang der einzelnen Abschnitte nicht vorgeschrieben. Die Gesamtfläche ist jedoch durch eine DIN A3 Seite begrenzt, das heißt der Umfang eines einzelnen Abschnitts kann zwar vergrößert werden, dies ist jedoch nur auf Kosten der anderen Abschnitte möglich.

Die Rückseite einer A3 Architektur-Übersicht enthält die visuelle Beschreibung des jeweiligen Architekturaspekts mit Hilfe von graphischen Modellen. Diese Seite besteht aus folgenden Abschnitten:

- Legende
- Funktionale Sicht
- Visuelle Unterstützung (der Verständlichkeit der funktionalen Sicht)
- Physikalische Sicht
- Parameterdefinition
- Einschränkungen des Entwurfs und Entwurfsentscheidungen

Auch auf dieser Seite der Architekturbeschreibung gilt, dass eine Abbildung nur auf Kosten der anderen Abbildungen vergrößert werden kann. Die einzelnen Abbildungen können jedoch verknüpft werden, um beispielsweise Verbindungen oder Abhängigkeiten zwischen diesen darzustellen.

Für die A3 Architektur-Übersichten ist zwar festgelegt, welche grundsätzlichen Aspekte oder Sichtweisen auf ein System beschrieben werden sollen, wie diese Beschreibung im Detail auszusehen hat, bleibt jedoch offen. So können für jeden Anwendungsfall die passenden Beschreibungsformen selbst gewählt werden.

### 2.3.3 System kohärenter Partialmodelle

In Paderborn wurde im Rahmen des SFB 614 „Selbstoptimierende Systeme des Maschinenbaus“ eine Spezifikationstechnik zur Beschreibung der Prinziplösung von selbstoptimierenden mechatronischen Systemen entwickelt [GFDK09]. Dabei werden mit einem System aus acht kohärenten Partialmodellen die wesentlichen Aspekte eines mechatronischen Systems beschrieben. Diese Aspekte sind die Anforderungen, die Anwendungsszenarien, die Funktionen, das Verhalten, die Gestalt, die Wirkstruktur, das Zielsystem und die Umgebung. Da die einzelnen Partialmodelle einige Querverbindungen enthalten und voneinander abhängig sind, bilden sie ein System kohärenter Modelle. Damit enthält das System der Partialmodelle mehrere Modelle, die parallel verändert werden, und entsprechende Abhängigkeiten besitzen. Es eignet sich somit als gutes Beispiel zur Anwendung des in dieser Arbeit vorgestellten Konzepts und die einzelnen Modelle werden im Folgenden detaillierter vorgestellt.

### Anforderungen

In diesem Partialmodell werden die Anforderungen an das zu entwickelnde mechatronische System modelliert. Die Anforderungen werden in einer Liste gesammelt und dort sowohl textuell beschrieben als auch durch Eigenschaften oder Zahlenwerte formalisiert hinterlegt. Sie werden als Forderung (muss erfüllt werden) oder Wunsch (sollte nach Möglichkeit erfüllt werden) gekennzeichnet [FeGr13]. Anforderungen können beispielsweise Leistungsanforderungen, Bauraum- oder Kostenbeschränkungen oder Bedienbarkeitsanforderungen sein.

### Anwendungsszenarien

Anwendungsszenarien beschreiben spezielle Ausschnitte der Funktionalität und mögliche Abläufe. Mit ihrer Hilfe werden die Reaktionen des Systems auf verschiedene Situationen oder Ereignisse spezifiziert und zum Beispiel festgelegt, welche Zustandsübergänge in den einzelnen Fällen stattfinden sollen. Die Spezifikation erfolgt in einer textuellen Beschreibung, die durch Graphiken unterstützt werden.

### Funktionen

Das Partialmodell Funktionen beschreibt die grundsätzliche Funktionalität des Systems. Hierbei werden die einzelnen Funktionen, analog zu den funktionsbasierten Ansätzen, hierarchisch gegliedert und so lange verfeinert, bis sie mit Hilfe von Lösungsprinzipien (für Elementarfunktionen [Koll98]) erfüllt werden können. Funktionen können einerseits herkömmliche Funktionen sein, die den Zusammenhang zwischen Eingangsgrößen und Ausgangsgrößen beschreiben, oder die für diesen Ansatz speziellen Selbstoptimierungsfunktionen definieren.

### Verhalten

Das Verhalten des zu entwickelnden Systems wird in einer Gruppe von Modellen beschrieben, da ein einzelnes Modell in der Regel nicht ausreichend ist. Beschrieben werden die Systemzustände mit den zugehörigen Ablaufprozessen und die Zustandsübergänge mit den zugehörigen Anpassungsprozessen (Selbstoptimierungsprozesse). Werden vernetzte Systeme entwickelt, so ist zusätzlich die Interaktion der einzelnen Teilsysteme zu definieren. Des Weiteren kann es erforderlich sein, dass die Kinematik oder Dynamik eines Systems explizit beschrieben wird. Dieses Partialmodell enthält somit Beschreibungen, die in Form von Zustands-, Aktivitäts- und Sequenzdiagrammen realisiert sind.

### Gestalt

Die Gestalt beschreibt die grobe physikalische Anordnung der einzelnen Systemelemente. Sie dient dem gemeinsamen Verständnis, um auch bereits in den frühen Entwicklungsphasen eine ungefähre dreidimensionale Vorstellung des späteren Systems zu haben. So können die Wirkflächen und die Wirkorte sowie Anzahl, Form und Lage der Systemelemente entnommen werden.

Dieses Modell besteht also aus der dreidimensionalen Anordnung von Volumenkörpern, deren Oberflächen gegebenenfalls mit Annotationen versehen werden können.

### Wirkstruktur

Im Partialmodell Wirkstruktur werden die Systemelemente sowie deren Eigenschaften und Beziehungen untereinander dargestellt, um die Struktur des zu entwickelnden Systems zu beschreiben. Systemelemente können hierarchisch angeordnet sowie in sogenannten „logischen Gruppen“ zusammengefasst werden, um die Übersichtlichkeit zu gewährleisten [GaKa10]. Stoff-, Energie- und Informationsflüsse zwischen den Systemelementen inklusive deren Richtung werden mit Hilfe von Pfeilen dargestellt und gemäß dem Entwicklungsfortschritt detailliert. Zudem enthält das Modell Verknüpfungen zu den anderen Partialmodellen, um beispielsweise erfasste Messgrößen oder den Einfluss von Störgrößen darzustellen.

### Zielsystem

Das Partialmodell Zielsystem beschreibt die Optimierungsziele des zu entwickelnden selbstoptimierenden Systems. Um Verwechslungen mit dem Begriff des Zielsystems der Embedded-Softwareentwicklung (Mikrocontroller, auf dem die in der Regel auf einem PC entwickelte Software ausgeführt werden soll) zu vermeiden, wird im Folgenden vom System der Ziele gesprochen. In diesem Partialmodell werden die Ziele des Systems hierarchisch aufgegliedert und in die Bereiche externe Ziele (z. B. Ziele der späteren Benutzer), inhärente Ziele (grundlegend, z. B. Sicherheit oder Wartung betreffend) und interne Ziele (Optimierungsziele des Systems) gruppiert. Ebenso werden die Abhängigkeiten der Ziele aufgezeigt, um gegenseitige Beeinflussungen bereits frühzeitig erkennen und berücksichtigen zu können.

### Umfeld

Die Umgebung des zu entwickelnden Systems wird im Partialmodell Umfeld beschrieben. Zum Umfeld gehört die Umwelt, das heißt Umwelteinflüsse wie Temperatur, Schmutz oder Feuchtigkeit, die das zu entwickelnde System beeinflussen können. Zum Umfeld gehören aber auch benachbarte Systeme oder Objekte, die entweder einen (passiven) Einfluss auf das System besitzen oder mit denen das System interagiert. Die auftretenden Einflüsse oder Störgrößen werden in diesem Modell inklusive deren Wechselwirkungen beschrieben, um beispielsweise im Partialmodell Verhalten eine entsprechende Reaktionen definieren zu können.

Die beschriebenen Partialmodelle bilden ein kohärentes System und können deshalb nicht unabhängig voneinander erstellt werden. Bei der Entwicklung eines mechatronischen Systems werden diese Modelle also iterativ immer wieder ergänzt und somit quasiparallel erstellt. Erst wenn das System der Partialmodelle konsistent ist und alle Partialmodelle einen ausreichenden Detaillierungsgrad aufweisen, um das zu entwickelnde System ausreichend zu beschreiben, kann mit der domänenspezifischen Entwicklung fortgefahren werden. Aber auch in dieser späteren Ent-

wurfsphase muss das System der Partialmodelle immer wieder an Änderungen angepasst werden, da es als gemeinsame disziplinübergreifende Referenz genutzt wird.

In diesem Kapitel wurde die Mechatronik als Vereinigung der Disziplinen Maschinenbau, Elektrotechnik und Informationstechnik vorgestellt. Diese Multidisziplinarität muss bei der Entwicklung berücksichtigt werden und spezielle Modellierungsansätze sind erforderlich. Bis auf das System der kohärenten Partialmodelle bleiben die Modellbeschreibungen jedoch relativ vage. Den Entwicklern bleiben so verhältnismäßig viele Freiheitsgrade bei der Definition bzw. Auswahl der zu verwendenden Modelle. Dies führt zu individuellen Modellierungskonzepten und zu einer Heterogenität in der Modelllandschaft, welche für eine modellübergreifende Konsistenzprüfung berücksichtigt werden muss. Dazu ist ein gewisses Maß an Flexibilität seitens eines Prüfungssystems erforderlich, die beispielsweise durch Softwareagenten realisiert werden kann. Diese werden im folgenden Kapitel vorgestellt.

# 3 Grundlagen zu Softwareagenten

Der Grundgedanke der agentenorientierten Entwicklung ist, dass ein System aus einer Gruppe von „intelligenten“ Agenten besteht, die gemeinsam bzw. in Kooperation das Systemziel erfüllen. Um diese abstrakte Vorstellung und die sich dadurch ergebenden Möglichkeiten besser zu verstehen, ist es notwendig, die dahinter stehende Idee genauer zu betrachten. Dazu wird zunächst eine Einführung in die agentenorientierten Softwareentwicklung gegeben, die das Vorgehen bei der Entwicklung von Agentensystemen beinhaltet. Anschließend werden die Konzepte, die in einem Agenten umgesetzt sind bzw. umgesetzt sein können, vorgestellt. Der Aufbau eines einzelnen Agenten und eines Agentensystems, häufig basierend auf einer Agentenplattform, bieten einen tieferen Einblick in die Realisierung solcher Systeme. Schließlich werden die vielfältigen Einsatzmöglichkeiten von Agentensystemen aufgezeigt. Zunächst soll jedoch geklärt werden, wie die Agenten entstanden sind und warum sie mittlerweile so nutzbringend eingesetzt werden.

## 3.1 Einführung in die agentenorientierte Softwareentwicklung

In den achtziger Jahren wurde in der Informatik im Bereich der Künstlichen Intelligenz ein neuer Forschungszweig gegründet, der sich mit Agentensystemen beschäftigt. Zu dieser Zeit verstand man darunter vorwiegend verteilte Problemlösungsprozesse [WGU03]. Spätestens seit den Arbeiten von Wooldridge und Jennings [WoJe95, JeWo01] haben Agenten ihr Schattendasein verlassen und erhalten Einzug in neue Bereiche außerhalb der Informatik. So werden sie beispielsweise im E-Commerce [CRR99] oder in der Logistik [KuGe13] eingesetzt. Speziell in der Automatisierungstechnik sind Agentensysteme auf dem Vormarsch und finden immer häufigeren Einsatz. Hier werden sie nicht nur in der direkten Produktionssteuerung wie zum Beispiel einer Fertigungsanlage für Zylinderköpfe in [SuBu01] sondern auch in begleitenden Prozessen wie der Planung für die Pfannenbelegung in einem Stahlwerk [VDI2653c] eingesetzt. Weitere und detailliertere Beispiele finden sich in Kapitel 3.5.

Die Anwendung von Agentensystemen eignet sich nach [VDI2653a] vor allem bei Problemstellungen mit verteilter, jedoch zusammenhängender Funktionalität, bei struktureller Veränderbarkeit, bei komplexen und variablen Abläufen sowie bei umfangreichen Koordinationsprozessen. Warum dies so ist, erkennt man, wenn man Agentensysteme näher betrachtet.

Ein Agentensystem besteht aus einer Menge von autonomen (Software-)Einheiten, den Agenten. Diese arbeiten selbstständig, können miteinander kommunizieren und kooperieren. Es existieren mehrere Versuche, einen Agenten zu definieren [Weiß01]. Eine Definition findet sich in [VDI2653a]:

**(technischer) Agent:**

*„Ein Agent ist eine abgrenzbare (Hardware- oder/und Software-) Einheit mit definierten Zielen. Ein Agent ist bestrebt, diese Ziele durch selbstständiges Verhalten zu erreichen und interagiert dabei mit seiner Umgebung und anderen Agenten.“ [VDI2653a]*

Die vorliegende Arbeit bezieht sich auf Softwareagenten, die im Folgenden vereinfacht als Agenten bezeichnet werden. Trotzdem sind viele Passagen in diesem Kapitel auch für Hardware- und Hardware/Softwareagenten zutreffend.

Agenten arbeiten einerseits autonom, sind andererseits jedoch in der Lage, miteinander zu kommunizieren. Dadurch eignen sie sich, verteilte Problemstellungen zu lösen, die beispielsweise nicht oder nur mit großem Aufwand zentral gelöst werden können. Dazu gehören Koordinationsaufgaben, die von den Agenten auch über Systemgrenzen hinweg bearbeitet werden können. Agenten kommunizieren über Protokolle und sind somit in der Lage auf relativ hohem Abstraktionsniveau Verhandlungen zu führen. Standards, wie sie zum Beispiel von der FIPA (Foundation for Intelligent Physical Agents) veröffentlicht werden, sichern den einheitlichen Aufbau der Nachrichten [FIPA02]. Das ermöglicht eine lose Kopplung der Agenten und erhöht damit die Flexibilität der Struktur von Agentensystemen. Es wird selbst eine Kommunikation zwischen Agenten heterogener Systeme realisierbar. Die lose Kopplung gemeinsam mit der autonomen und selbstständigen Arbeitsweise der Agenten vereinfacht die Entwicklung eines Agentensystems dahingehend, dass nicht alle möglichen Abläufe bereits bei der Entwicklung festgelegt werden müssen. Diese ergeben sich zur Laufzeit aus dem dynamischen Zusammenspiel der Agenten, die innerhalb des ihnen zugewiesenen Handlungsspielraums frei agieren und reagieren können. Somit bleiben auch komplexere Problemstellungen beherrschbar, indem die Komplexität auf mehrere Agenten verteilt wird.

### Entwicklung von Agentensystemen

Dabei ist zu beachten, dass die Agentenorientierung zunächst keine grundsätzlich neue Art der Softwareerstellung bzw. -programmierung darstellt. Das Paradigma der agentenorientierten Softwareentwicklung (AOSE) ist vielmehr eine neue Herangehensweise an die Softwareentwicklung und den Entwicklungsprozess, so wie es die Objektorientierung im Vergleich zur prozeduralen Softwareentwicklung war. Die AOSE ist zu einem wirkungsvollen Hilfsmittel für die Entwicklung von flexiblen Systemen für umfangreiche und komplexe Aufgabenstellungen geworden [Jenn00].

Zur Entwicklung eines Agentensystems gibt es zahlreiche Methoden, nach denen vorgegangen werden kann und die verschiedene Entwurfsphasen abdecken. Verbreitete Beispiele sind Gaia [WJK00], PASSI (Process for Agent Societies Specification and Implementation) [Coss05] oder MaSE (Multi-agent Systems Engineering) [WoLe01]. Für weitere Beispiele sei unter anderem

auf [VDI2653b] verwiesen. Gaia deckt die Analyse und den groben Entwurf ab, PASSI und MaSE eignen sich für den gesamten Entwurfsprozess.

Alle Methoden schlagen ein grundsätzlich ähnliches Vorgehen vor. Zunächst werden an Hand der Anforderungen Ziele und/oder Rollen bzw. Aufgaben für die späteren Agenten abgeleitet. Anschließend werden diese zu Agententypen zusammengefasst und die notwendigen Kommunikationspfade zwischen diesen bestimmt. Schließlich werden der Aufbau der Agenten sowie die Kommunikationsprotokolle genauer spezifiziert. Am Ende wird das gesamte System implementiert. Je nach Methode wird dieses Vorgehen in unterschiedlich viele Phasen unterteilt und mit diversen Modellen unterstützt.

In einigen Fällen kann es sinnvoll und notwendig sein, eine der etablierten Methoden an den speziellen Anwendungsfall anzupassen, um alle Anforderungen abdecken zu können. Beispiel hierfür ist die Erweiterung der Methode MaSE um die Berücksichtigung von Echtzeitanforderungen in eingebetteten Systemen, die in [BMG08] vorgestellt wird.

## 3.2 Agentenorientierte Konzepte

Bei der Entwicklung eines Agentensystems kommen im einzelnen Agenten verschiedene Konzepte zum Einsatz (siehe Abbildung 3.1).


Abbildung 3.1: Agentenkonzepte nach [WGU03]

Diese Konzepte treten, je nach dem für welchen Anwendungsfall das Agentensystem entwickelt wurde, mehr oder weniger stark hervor:

- *Kapselung*: Ein Agent kapselt seinen inneren Zustand ähnlich wie ein Objekt in der Objektorientierung. Zudem kapselt er sein Verhalten und dieses ist nach außen nicht sichtbar.

- *Zielorientierung*: Ein Agent besitzt ein oder mehrere Ziele, die er verfolgt und die sein Verhalten beeinflussen. Diese Ziele können bereits bei der Entwicklung festgelegt oder später zur Laufzeit vom Agenten selbst oder von außen zugeordnet bzw. geändert werden.

- *Aktivität*: Ein Agent besitzt ein aktives Verhalten und muss nicht von außen angestoßen werden. Er kann somit aktiv auf Veränderungen in seiner Umwelt reagieren. Zusammen mit der Zielorientierung ist auch ein proaktives (vorausschauendes) Verhalten möglich.

- *Persistenz*: Ein Agent besitzt einen eigenen fortlaufenden Kontrollfluss, der unabhängig von einer Aktivierung von außen ist.

- *Interaktion*: Ein Agent kann mit seiner Umwelt, dazu gehören auch andere Agenten, interagieren. Die Kommunikation wird über Protokolle definiert und kann auf einem hohen Abstraktionsgrad stattfinden. So können Agenten beispielsweise miteinander verhandeln oder ihr Vorgehen zum Erreichen ihrer jeweiligen Ziele abstimmen.

- *Autonomie*: Ein Agent besitzt die Kontrolle über seinen inneren Zustand und sein Verhalten. Das heißt er entscheidet an Hand seines Wissens selbst, ob und wie er zum Beispiel auf eine Veränderung in der Umgebung reagiert oder wie er seine Ziele verfolgt.

Zusätzlich können Agenten in gewissen Anwendungsgebieten die folgenden Eigenschaften aufweisen:

- *Anpassungsfähigkeit/Lernfähigkeit*: Ein Agent kann sich auf Grund seiner Erfahrungen an seine Umgebung anpassen und sein Verhalten ändern. Er kann somit selbstständig dazulernen und sich verbessern.

- *Mobilität*: Ein Agent kann mobil sein, sich von Plattform zu Plattform oder von System zu System bewegen und so seinen Ausführungsort wechseln.

Die beiden letztgenannten Eigenschaften wurden lange Zeit eher selten benötigt, nehmen aber durch neue Anforderungen an entsprechende Systeme und begünstigt durch neue Realisierungsmöglichkeiten an Wichtigkeit zu.

## 3.3 Aufbau eines Agenten

Der innere Aufbau eines Agenten, also dessen Architektur, ist entscheidend, um die oben genannten Konzepte zu realisieren. Dabei gibt es verschiedene Ausbaustufen, die sich in ihrer Komplexität unterscheiden und sich sehr auf die Eigenschaften des Agenten auswirken.

Der am einfachsten aufgebaute Agent ist der reaktive Agent. Er reagiert direkt auf Änderungen in seiner Umgebung, was beispielsweise durch ein Regelwerk realisiert werden kann. Dabei besitzt der Agent kein Gedächtnis, das heißt Historiendaten werden nicht berücksichtigt

[SCT05]. Der Vorteil von reaktiven Agenten ist die höhere Reaktionsgeschwindigkeit gegenüber deliberativen Agenten, da lediglich ein Satz von Regeln ausgewertet werden muss.

Deliberative (zu Deutsch: abwägende) Agenten sind mächtiger, allerdings auch komplexer aufgebaut. Sie setzen das Konzept der Planung um und ermöglichen so eine proaktive Verhaltensweise. Eine weit verbreitete Form der deliberativen Agenten ist der BDI-Agent (Belief-Desire-Intention) [BJW04]. Der Agent besitzt in diesem Fall ein gewisses Verständnis und Abbild seiner Umgebung bzw. ein Weltbild (Belief). Er besitzt Ziele (Desire), die er erreichen möchte und er besitzt eine Handlungsabsicht (Intention). Die Handlungsabsicht beschreibt die Tätigkeiten oder Aktionen, die der Agent ausführt, um seine Ziele zu erreichen, und kann treffender als dessen Verhalten bezeichnet werden.

Deliberative Agenten können zudem mit der Fähigkeit zu Lernen ausgestattet werden. Sie sind somit in der Lage, ihr internes Wissen (Weltbild und/oder Fähigkeiten) zum Beispiel auf Grund der von ihnen gemachten Erfahrungen selbstständig zu erweitern.

Da rein deliberative Agenten im Vergleich zu reaktiven Agenten eher träge sind (komplexer Planungsablauf), werden häufig Mischformen, sogenannte hybride Agenten eingesetzt. Diese sind in der Regel mehrschichtig aufgebaut und vereinen die hohe Reaktionsgeschwindigkeit der reaktiven Agenten mit der abstrakten Planungsfähigkeit der deliberativen Agenten [BJW04].

Eine Darstellung des inneren Aufbaus eines Agenten wird in [GUW04] vorgestellt (siehe Abbildung 3.2). An Hand dieser Darstellung können sowohl reaktive als auch deliberative Agenten beschrieben werden, wobei die einzelnen Elemente jeweils mehr oder weniger stark ausgeprägt sind.

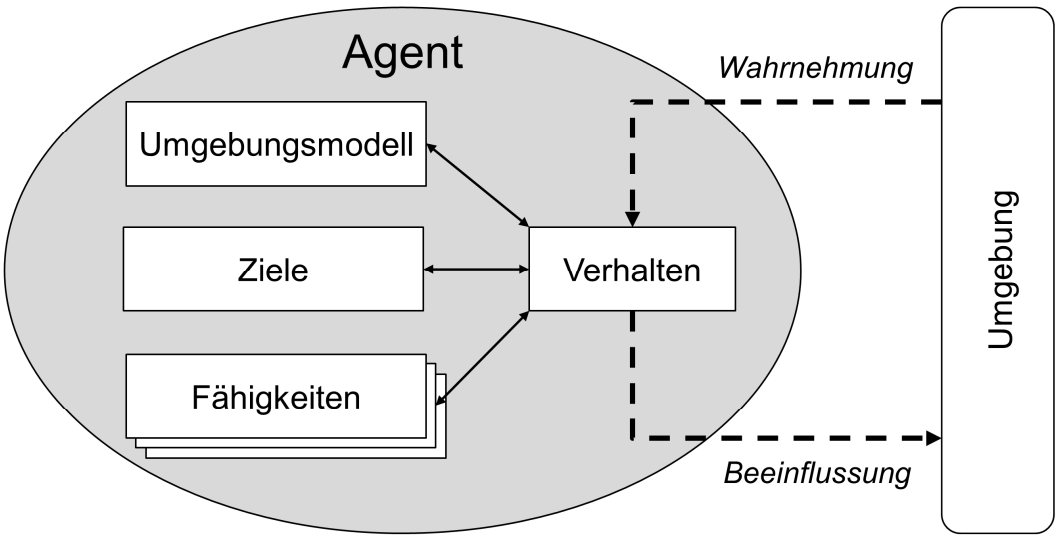


Abbildung 3.2: Innerer Aufbau eines BDI-Agenten nach [GUW04]

Der Agent nimmt seine Umgebung wahr, was so viel bedeutet, dass er diese auf Grund seines Verhaltens beobachtet und/oder Nachrichten von anderen Agenten empfängt. Mit den erlangten

Informationen wird das Umgebungsmodell (Weltbild) aktualisiert oder erweitert (bei lernenden Agenten). Dieses hat wiederum, gemeinsam mit den Zielen, Einfluss auf das Verhalten des Agenten. Der Agent selbst wurde von seinem Entwickler mit gewissen Fähigkeiten ausgestattet, die er benötigt, um seine Ziele zu erreichen. Aus diesen Fähigkeiten wird das Verhalten zusammengesetzt. Darunter können Fähigkeiten sein, die es dem Agenten ermöglichen, seine Umgebung zu beobachten oder Nachrichten von anderen Agenten zu empfangen. Es können aber auch Fähigkeiten zu Berechnung komplexer Zusammenhänge, der Planung von Handlungen oder das Erlernen neuer Fähigkeiten sein. Im Rahmen seiner Fähigkeiten beeinflusst der Agent schließlich seine Umgebung, um seine Ziele zu erreichen.

## 3.4 Aufbau eines Agentensystems

Zur Umgebung eines Agenten gehören auch die anderen Agenten im Agentensystem. Damit diese innerhalb des Systems zusammenarbeiten und miteinander kommunizieren können, muss eine gewisse Infrastruktur vorhanden sein. Die erforderlichen Funktionalitäten, dazu gehören Nachrichtenaustausch, Agentenverwaltung und Verzeichnisdienste, können je nach Einsatzzweck des Agentensystems mehr oder weniger wichtig sein und müssen daher nicht in allen Fällen vollständig vorhanden sein. An Hand der FIPA Agent Management Specification [FIPA04] werden die Funktionalitäten im Folgenden vorgestellt (vgl. Abbildung 3.3).

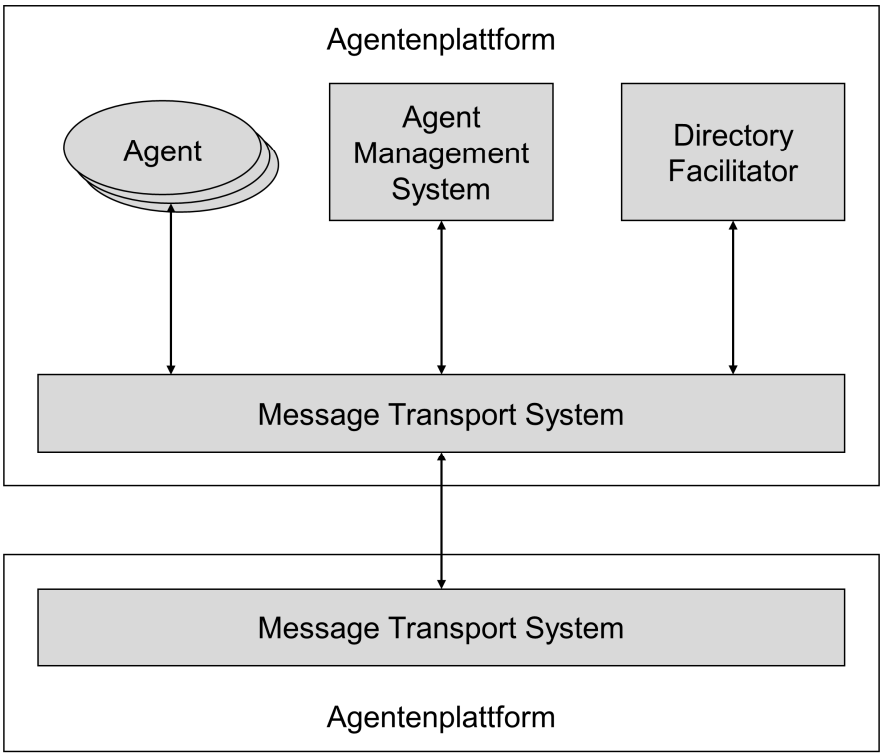


Abbildung 3.3: Agentenplattform nach [FIPA04]

Die Kommunikationsinfrastruktur eines Agentensystems bzw. der Agentenplattform ist das *Message Transport System*. Es sorgt dafür, dass die Nachrichten zwischen den Agenten, auch

über Systemgrenzen hinweg, weitestgehend transparent zugestellt werden. Eine Nachricht besteht dabei aus einem Umschlag (Envelope), der unter anderem den oder die Empfänger sowie den Absender enthält. Der Nutzteil (Payload) stellt die Nutzdaten oder den Inhalt der Nachricht dar.

Das *Agent Management System* (AMS) verwaltet eine Agentenplattform. Es existiert nur einmal pro Plattform und vergibt die jeweils eindeutigen IDs an die Agenten. Zudem enthält es das Agentenverzeichnis (white pages). Jeder Agent einer Plattform muss sich beim AMS registrieren, wenn er instanziiert wird, und abmelden, wenn er aufgelöst wird.

Der *Directory Facilitator* (DF) ist für eine Agentenplattform nicht zwingend erforderlich, in vielen Fällen jedoch sehr nützlich. Er bietet einen Verzeichnisdienst, die sogenannten Gelben Seiten (yellow pages). Die Agenten können sich beim DF mit ihren Diensten bzw. Fähigkeiten (z. B. bestimmte Informationen bereitzustellen) registrieren. Andere Agenten können in den Gelben Seiten suchen und sind somit in der Lage, Anbieter von gewünschten Diensten zu finden.

In vielen Fällen werden diese Funktionalitäten in Form einer Agentenplattform, die eine Art Laufzeitumgebung für die Agenten darstellt, bereitgestellt [VDI2653b]. Prinzipiell kann bei der Implementierung eines Agentensystems auch eine komplett neue Entwicklung begonnen werden oder die Funktionalitäten können direkt in die Agenten integriert werden. Bei der Verwendung einer bereits verfügbaren Agentenplattform reduziert sich der Implementierungsaufwand eines Agentensystems jedoch beträchtlich. Obwohl bereits Agentenplattformen für eingebettete Systeme existieren, kann es beispielsweise bei verteilten Systemen sinnvoll sein, auf keine der verfügbaren Plattform zurückzugreifen, da die Laufzeitumgebung mit den angebotenen Diensten oft einen verhältnismäßig großen Ressourcenverbrauch besitzt und nicht immer alle Dienste benötigt werden.

### Agentenplattform JADE

Eine verbreitete Agentenplattform ist JADE (Java Agent Development Framework) [JADE10]. JADE ist jedoch nicht nur eine Agentenplattform sondern stellt auch ein Implementierungsframework bereit. Es existieren beispielsweise vorgefertigte Rahmenkonstruktionen für Agenten, die bei der Implementierung lediglich mit Inhalt versehen werden müssen, und es werden Schnittstellen angeboten, mit deren Hilfe die Funktionalität individuell erweitert werden kann. Das Framework ist in der Programmiersprache Java implementiert und setzt die Standards der FIPA [z. B. FIPA02] um. In JADE besitzt ein Agent sogenannte Behaviours, die er zyklisch abarbeitet, und in denen das Verhalten des Agenten implementiert werden muss. Da einem Agenten Behaviours dynamisch zur Laufzeit hinzugefügt oder entfernt werden können, kann dessen Verhalten relativ einfach angepasst werden. Ebenso sind der Implementierung des Weltbilds des Agenten in der objektorientierten Programmiersprache Java nahezu keine Grenzen gesetzt. Lediglich bei der Geschwindigkeit müssen Abstriche gemacht werden, da Java-

Programme in der Regel in einer Laufzeitumgebung (Java-Runtime-Environment) und nicht direkt ausgeführt werden. Dafür gewinnt man die Plattformunabhängigkeit und Teile eines verteilten JADE-Agentensystems können beispielsweise auf einem Windowsbetriebssystem ausgeführt werden, während andere Teile in einer Linux-Umgebung laufen. JADE ist frei verfügbar und wird deshalb gerne in der Forschung eingesetzt.

## 3.5 Einsatzgebiete von Softwareagenten in der Automatisierungstechnik

Die Forschung beschäftigt sich in vielen Bereichen mit Agentensystemen und hat bereits einige Konzepte erstellt, die beschreiben wie Agenten in der Praxis zukünftig gewinnbringend eingesetzt werden können. In den folgenden Abschnitten werden ausgewählte Konzepte vorgestellt, um die Möglichkeiten des Einsatzes von Agentensystemen sowie deren Eignung zur Konsistenzprüfung von heterogenen Modellen zu verdeutlichen.

Grundsätzlich können Agentensysteme sowohl im produktiven Betrieb als auch im planerischen Bereich eingesetzt werden. Zum produktiven Betrieb gehören Agentensysteme, die beispielsweise eine Produktionsanlage steuern und damit direkt mit Sensoren und Aktoren gekoppelt sind. Auf Grund der Anbindung dieser Systeme an die reale Welt (technischer Prozess), müssen diese unter anderem echtzeitfähig sein [LaGö99]. Häufig müssen diese Systeme auf der Ebene der Feldgeräte arbeiten, d.h. es steht in der Regel kein mit ausreichend Ressourcen ausgestatteter PC mit einer komfortablen Laufzeitumgebung für die Agenten zur Verfügung. Agenten werden in diesen Anwendungsfällen also beispielsweise auf Mikrocontrollern [PeGö10] oder speicherprogrammierbaren Steuerungen (SPS) [Wann10] eingesetzt.

Ein deutlich größeres Anwendungsspektrum gibt es für Agentensysteme im planerischen Bereich. Hier muss in der Regel keine direkte und echtzeitfähige Kopplung an die Realität berücksichtigt werden und die Bandbreite für den Einsatz von Agentensystemen erstreckt sich von Produktionsplanungsaufgaben über Simulationen bis hin zu Assistenzsystemen beim Entwurf.

Die Planung und Optimierung des Fertigungsablaufs für flexible Fertigungssysteme wird in [Badr11] beispielsweise mit Hilfe eines Agentensystems durchgeführt. Dabei kennen die Agenten die Randbedingungen (z. B. Anzahl, Fähigkeiten und Kapazitäten der Fertigungseinrichtungen) und reagieren selbstständig auf neu eingehende Produktionsaufträge mit unterschiedlichen Prioritäten oder Maschinenausfälle, indem sie den Fertigungsablauf anpassen.

Im Forschungsprojekt EffiPro (Effizienzsteigerung in der Produktion durch agentenbasierte Fertigungssystemplanung) wird die Konfigurationsplanung für rekonfigurierbare Werkzeugmaschinen von Agenten durchgeführt [HGB+10, HGBR14]. Einzelne Entitäten wie Fertigungsvorgänge, Werkzeuge oder Bearbeitungsmodule werden durch Agenten repräsentiert und aufeinander abgestimmt.

[Malz13] beschreibt ein Konzept, in dem die Priorisierung von Testfällen beim Softwaretest mit Hilfe eines Agentensystems ausgeführt wird. Die Agenten besitzen Informationen über die Testfälle, die zum Testen benötigte Ressourcen und die zu testenden Softwaremodule. Diese Informationen werden entsprechend bestimmter Regeln verknüpft und Formeln berechnet, um die Priorität zu bestimmen. Dazu ist es notwendig, verschiedene bestehende Datenquellen in das System einzubinden. Um diese Einbindung flexibel bezüglich des Wechsels der Datenquellenstruktur zu halten, werden Agenten als Schnittstellen eingesetzt. So muss bei einer Änderung der Datenquelle lediglich der Schnittstellenagent angepasst werden.

Häufig werden Agentensysteme für Simulationen verschiedener Art eingesetzt [MaNo10]. In [JBR+12, JBR+13] wird die Effizienz von verteilten numerischen Simulationen durch den Einsatz von Agenten gesteigert. Die Agenten verwalten jeweils eine Teilsimulation, gleichen deren Ergebnisse über Rechnergrenzen hinweg ab und passen die Simulationen gegebenenfalls an neue Randbedingungen an. Dabei laufen die einzelnen Teilsimulationen autonom ab und die zugehörigen Agenten kommunizieren über ein Netzwerk.

[Wagn08] beschreibt ein Assistenzsystem, das das Engineering von Automatisierungsanlagen unterstützt. Beim Engineering dieser Anlagen sind, ähnlich wie bei der Mechatronik, mehrere Disziplinen, hier als Gewerke (z. B. Verfahrenstechnik, Energietechnik etc.) bezeichnet, beteiligt. Jedes Gewerk besitzt eine eigene Sichtweise auf eine Anlage und für die Konsistenz des Anlagenentwurfs müssen gewerkspezifische Regeln eingehalten werden. Im Vergleich zur Mechatronik wird das Engineering jedoch auf einer höheren Abstraktionsebene durchgeführt. [Wagn08] geht davon aus, dass eine Anlage aus vordefinierten Komponenten zusammengesetzt wird, deren Eigenschaften über verschiedene Parameter (z. B. Pumpenleistung, Länge usw.) angepasst werden können. Jede Komponente der geplanten Anlage wird durch einen Komponenten-Agenten vertreten. Dieser Agententyp besitzt Informationen zur Konfiguration seiner Komponente, weiß mit welchen anderen Komponenten diese verknüpft ist, und kennt die Parametrier- und Verbindungsregeln für die Komponente. Er hat die Aufgabe Änderungen an der Konfiguration der Komponenten durch den Anlagenplaner zu erkennen und die Konsistenz der Komponenten sowie deren Verknüpfungen sicherzustellen. Eine Baugruppe, zusammengesetzt aus einzelnen Komponenten, wird ebenfalls als Komponente betrachtet. Ein Bibliotheks-Agent verwaltet eine Komponentenbibliothek und ist in der Lage, passende Komponenten zu suchen und zu instanziieren. Aspekt-Agenten vertreten schließlich die verschiedenen Gewerke und sind in der Lage, gewerkspezifische Teilmodelle aus dem übergeordneten Anlagenmodell abzuleiten. Innerhalb dieser Teilmodelle sichern sie die Konsistenz und passen Komponenten gegebenenfalls an. Dazu besitzen sie entsprechendes Wissen über das repräsentierte Gewerk allgemein und spezielle Regeln, die beim vorliegenden Projekt beachtet werden müssen. Das Assistenzsystem ist somit in der Lage, Inkonsistenzen im Anlagenmodell zu erkennen, und unterstützt den Anlagenplaner geeignete Anpassungen zur Auflösung der Inkonsistenzen zu finden.

Ein ähnliches Ziel, jedoch auf dem Gebiet der mechanischen Konstruktion, besitzt das in [KRBG11a] vorgestellte Assistenzsystem. Das an ein CAD-System angekoppelte Assistenzsystem unterstützt den Konstrukteur in der Detaillierungsphase, indem es Inkonsistenzen in der Konstruktion aufdeckt, automatisiert Lösungsvorschläge erarbeitet und diese auf Wunsch des Konstrukteurs umsetzt. Dazu werden die einzelnen Konstruktionselemente sowie Verbindungen zwischen diesen und übergeordnete Baugruppen von Agenten (Bauteil-, Verbindungs- und Baugruppen-Agenten) vertreten. Diese erkennen Änderungen durch den Konstrukteur und initiieren eine Konsistenzprüfung. Ebenso stellen sie Informationen über das repräsentierte Element bereit und setzen gegebenenfalls erforderliche Anpassungen um. Ein Management-Agent entscheidet über die Durchführung einer Konsistenzprüfung und über die Erarbeitung von Lösungen. Die Prüfung selbst wird durch Aspekt-Agenten, die jeweils einen Konstruktionsaspekt (z. B. Funktion, Kosten, Werkstoff etc.) repräsentieren, ausgeführt. Diese besitzen Wissen in Form von Regeln, Formeln oder Tabellen aus verschiedenen Normen und sind in der Lage, dieses Wissen anzuwenden. Informationen über die Konstruktionselemente bekommen sie unter anderem von den Bauteil-Agenten. Unterstützt werden sie durch die Fach-Agenten, die Expertenwissen für bestimmte Konstruktionselemente (z. B. Welle oder Lager) besitzen. Gemeinsam mit den Aspekt-Agenten erarbeiten sie mit Hilfe eines verteilten Problemlösungsverfahren, das die Bauteil-Agenten als Blackboard benutzt, Anpassungen an der Konstruktion, um Inkonsistenzen aufzulösen und den Konstrukteur dadurch zu unterstützen und zu entlasten.

In diesem Kapitel wurden das Paradigma der agentenorientierten Softwareentwicklung als wirkungsvolles Werkzeug zur Lösung komplexer und verteilter Aufgabenstellungen vorgestellt. Es wurde gezeigt, wie Agentensysteme entwickelt werden und welche Konzepte einen Agenten ausmachen. Die lose Kopplung über Kommunikationsprotokolle ermöglicht beispielsweise die flexible Integration neuer Agenten zur Laufzeit und die Autonomie ein selbstständiges Verhalten der Agenten. Die verhältnismäßig einfachen reaktiven und die mächtigen deliberativen Agenten bieten viele Möglichkeiten für den Einsatz von Agentensystemen. Verfügbare Agentenframeworks erleichtern die Realisierung. In Forschung und Industrie finden sich daher viele Beispiele, in denen Agenten nutzbringend eingesetzt werden. Weitere Beispiele können unter anderem in [Göhn13] gefunden werden. Das Wissen, das Agenten benutzen, muss jedoch in verarbeitbarer Form im System vorliegen. Eine Möglichkeit eine entsprechende Wissensbasis zu realisieren sind Ontologien. Diese werden im folgenden Kapitel vorgestellt.

# 4 Grundlagen zu Ontologien

Obwohl der Begriff der Ontologie ursprünglich aus der Philosophie stammt, bezeichnet er heutzutage ein mächtiges Werkzeug, um Wissen im technischen Bereich abzubilden. In einer Einführung wird deshalb zunächst die Grundidee von Ontologien vorgestellt und es wird erläutert, was das Reasoning bedeutet. Anschließend folgt der Aufbau von Ontologien. Dazu gehören neben der Struktur auch die Sprachen, mit denen eine Ontologie formuliert wird. Andere Sprachen werden für die Abfrage von Wissen aus der Ontologie benutzt, weshalb der Vergleich mit einer Datenbank herangezogen wird. Die Erstellung von Ontologien besteht aus mehreren Teilschritten, von denen die Begriffsbildung und die Evaluierung detaillierter vorgestellt werden. Schließlich werden Anwendungsbeispiele von Ontologien im technischen Bereich vorgestellt. Zunächst muss jedoch geklärt werden, was Ontologien sind und warum sie eingesetzt werden.

## 4.1 Einführung in Ontologien

Heutige Computersysteme nehmen dem Menschen mehr und mehr Arbeit ab, da die (Software-) Systeme immer schneller, besser und „intelligenter“ werden. Das Wissen, dass der Mensch anwendet, um die gegebenen Problemstellungen zu lösen, muss dazu in die Software integriert werden. Ist das spezifische Fachwissen vollständig vom allgemeinen Problemlösungswissen getrennt, spricht man von einem Wissensbasierten System (WBS) [RFSE11].

Aus technischer Sicht gibt es verschiedene Möglichkeiten, Wissen in ein System zu integrieren. Formeln können beispielsweise relativ einfach, jedoch bezüglich Änderungen unflexibel direkt im Quellcode eines Programms hinterlegt werden. Besser ist es, das Wissen vom eigentlichen Quellcode weitgehend zu trennen und dieses in separaten Dateien oder Datenbanken zu hinterlegen. Dieses kann so erweitert werden, ohne dass der Quellcode geändert und die Software neu kompiliert werden muss. In diesem Kapitel soll jedoch zunächst die benötigte Form (abstrakte Darstellung) des Wissens untersucht werden.

Das zu integrierende Wissen kann von unterschiedlicher Komplexität sein und, je nach Art des Wissens, unterschiedlich dargestellt werden. Verhältnismäßig einfache Formen von Wissen sind Listen oder Tabellen. Hierbei existieren mehrere gleichartige Wissenselemente, die gegebenenfalls in eine Reihenfolge gebracht werden. So lassen sich beispielsweise Mengen oder Kennfelder darstellen. Die Komplexität steigt, wenn die Anzahl der Verknüpfungen zwischen den einzelnen Wissenselementen erhöht und variabel gemacht wird und verschiedenartige Verknüpfungen existieren. Das Ergebnis ist ein Netz aus Wissenselementen, deren Verknüpfungen unterschiedliche Bedeutungen besitzen. Eine einfache Form eines solchen Netzwerks ist eine Taxonomie (Hierarchie bzw. Baumstruktur), die allgemeine Form ist das Semantische Netzwerk (Netzstruktur).

Damit das Wissen von einer Software benutzt und interpretiert werden kann, muss es formalisiert werden. Ein solches formalisiertes semantisches Netzwerk wird als Ontologie bezeichnet [SaAl08]. Ursprünglich stammt der Begriff der Ontologie aus der Philosophie, wo er die „Lehre des Seins“ beschreibt [Grub93]. Er wird in der Informationsverarbeitung jedoch nicht einheitlich benutzt [CJB99]. Einerseits wird das in ein System integrierte Wissen selbst als Ontologie bezeichnet und beschreibt damit „was ist“ (nahe an der Definition der Philosophie), andererseits wird der Begriff für eine technische Repräsentationsform von Wissen benutzt. In diesem Kapitel wird zum besseren Verständnis zunächst auf die abstrakte Bedeutung der Ontologie eingegangen und anschließend mit der technischen Realisierung fortgefahren. In den folgenden Kapiteln ist mit dem Begriff immer die technische Repräsentation des Wissens gemeint.

Eine sehr bekannte Definition für den Begriff der Ontologie in der Informationsverarbeitung findet sich in [Grub93], die von [Bors97] später ergänzt wurde:

**Definition Ontologie:**

*“An ontology is a formal specification of a shared conceptualization.”* [Bors97]

Eine Ontologie beschreibt also einen Sachverhalt oder einen Wissensbereich in der Art, dass die Beschreibung für alle Beteiligten verständlich ist. In gewisser Weise kann man sogar von einem gemeinsamen Weltbild (oder einem Ausschnitt davon) sprechen. Die Spezifikation von Ontologien erfolgt formal und ermöglicht so die Verarbeitung innerhalb eines Softwaresystems. Die technische Realisierung ist bei dieser Definition jedoch offen.

Betrachtet man die Ontologien als Repräsentationsform von Wissen, beschreiben sie Begriffe und deren Relationen zueinander. Sie bestehen deshalb in der Regel aus einer Taxonomie, die eine Klassifikation der Begriffe (hierarchische Ordnung) darstellt. Diese Taxonomie wird um semantische Relationen und Axiome bzw. Regeln zum Aufbau der Ontologie erweitert [Hess14]. So enthält eine Ontologie nicht nur eine Datenstruktur, sondern auch den Bauplan dieser Struktur. Eine detaillierte Beschreibung des Aufbaus von Ontologien findet sich in Kapitel 4.2.

Zur Auswertung bzw. Interpretation einer Ontologie wird eine Auswertelogik (Inferenzmaschine) benötigt. Diese wird als Reasoner bezeichnet [DCTK11]. Ein Reasoner ist in der Lage, logische Schlüsse aus der Ontologie zu ziehen, indem er die semantischen Verknüpfungen sowie die Axiome interpretiert.

Wenn beispielsweise B eine Unterklasse von A ist und C eine Unterklasse von B, so kann daraus geschlossen werden, dass C auch eine Unterklasse von A sein muss (vgl. Abbildung 4.1).

Oft bieten Reasoner neben der Funktionalität zur Interpretation einer Otologie auch die Funktionalität die Ontologie auf Konsistenz zu prüfen. Beispiele für Reasoner sind unter anderem in [DCTK11] zu finden.

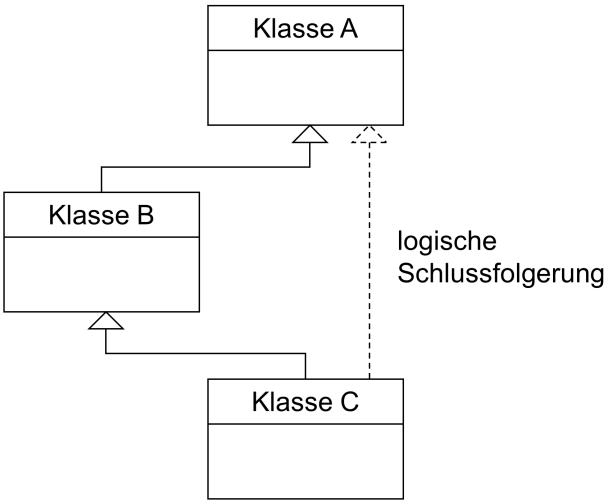


Abbildung 4.1: Beispiel für das Reasoning in einer Ontologie

Der Einsatz von Ontologien erstreckt sich von einem reinen Wissensspeicher für WBS bis hin zur Definition eines gemeinsamen Wortschatzes für Kommunikationszwecke. Häufig werden beide Anwendungsfälle genutzt, wobei der Schwerpunkt wechselt.

Durch die vielfältigen Einsatzmöglichkeiten finden Ontologien in die unterschiedlichsten Bereiche Einzug. Gemeinsamkeiten der verschiedenen Bereiche sind in der Regel die Notwendigkeit eines gemeinsamen Verständnisses eines Sachgebiets, die Verarbeitung und Bereitstellung von umfangreichem Wissen oder die Notwendigkeit einer gemeinsamen und eindeutigen Kommunikationsbasis. Beispiele für die ersten beiden Fälle finden sich in der Biologie [Blak13, WLT+06], der Medizin [Herr11], dem Engineering [MMWY10] oder im Bereich des E-Commerce und des Internets [DFK+04, Hepp08]. Für letzteren Fall gibt es Beispiele aus Kapitel 3 bekannten Agentensystemen [APCB02] oder aus der Landwirtschaft [GKB13].

## 4.2 Aufbau von Ontologien

Nachdem die Bedeutung von Ontologien dargelegt wurde, werden in diesem Unterkapitel der Aufbau und die technische Realisierung von Ontologien behandelt. Dazu gehören auch die Beschreibungssprachen, mit deren Hilfe Ontologien erstellt werden.

Wie bereits erwähnt, enthalten Ontologien Begriffe, Relationen und Axiome. Begriffe stellen die Knoten eines Graphen (semantisches Netz) dar und werden in einigen Quellen auch als Konzepte bezeichnet. Sie repräsentieren eine Entität des Wissens, das abgebildet werden soll. Beispiele für Begriffe sind *Auto*, *LKW* oder *Kraftfahrzeug*. Sie können Eigenschaften (manchmal als Parameter bezeichnet) aufweisen, die einen Begriff genauer beschreiben können. Diese Eigenschaften werden im Sinne der objektorientierten Softwareentwicklung vererbt [EhSt06]. Relationen sind Verknüpfungen zwischen Begriffen, die eine semantische Bedeutung erhalten. Ein Beispiel hierfür ist die *ist_ein*-Beziehung. So lassen sich aus Begriffen und Relationen Tripel bilden, die wiederum zu einem Netz zusammengesetzt werden können. Aus den Tripeln *Au-*

*to – ist_ein – Kraftfahrzeug* und *LKW – ist_ein – Kraftfahrzeug* lässt sich das in Abbildung 4.2 dargestellte Netz zusammensetzen.

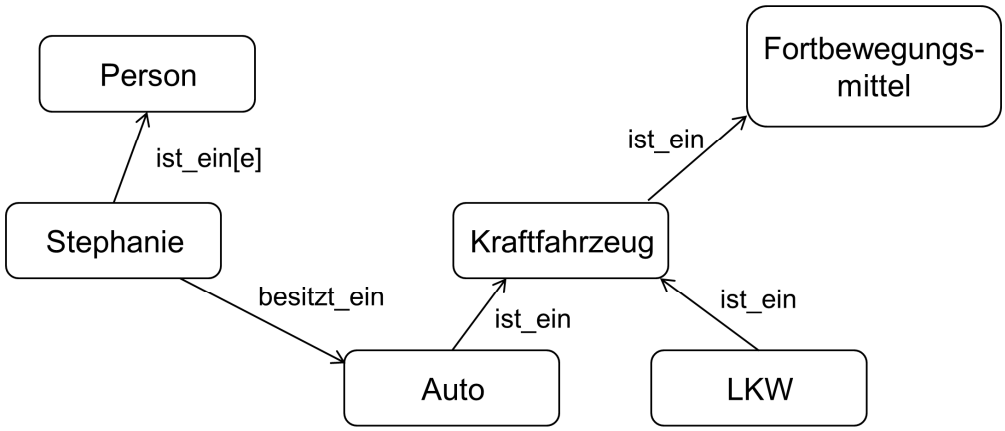


Abbildung 4.2: Einfaches Beispiel für ein Semantisches Netz

Axiome sind Informationen, die nicht ohne weiteres innerhalb des Semantischen Netzes abgebildet und damit daraus auch nicht abgeleitet werden können. Sie beschreiben Regeln, die beim Aufbau des Netzes eingehalten werden müssen, wie beispielsweise „Autos und LKWs sind nicht das gleiche Kraftfahrzeug“.

Die bisher beschriebenen Elemente einer Ontologie beschreiben die Konzept-Ebene der Ontologie (vgl. Abbildung 4.3), gelegentlich auch als TBox bezeichnet [DCTK11]. Sie ist bezüglich der Abstraktionsebene vergleichbar mit dem Klassendiagramm in der Objektorientierten Programmierung (OO) oder dem Entity-Relation-Diagramm der Datenbankentwicklung. Sie definiert den „Bauplan“ der Ontologie, der bei der Abbildung von Instanzen, vergleichbar mit Objekten in der OO, angewendet wird. Diese Konzept-Ebene ist die eigentliche Ontologie, da sie das Weltbild vorgibt. Häufig wird jedoch das Gesamtkonstrukt als Ontologie bezeichnet, was in dieser Arbeit zur Vereinfachung genauso gehandhabt wird.

Die Instanz-Ebene der Ontologie enthält die Instanzen, die unter Einhaltung der in der Konzept-Ebene definierten Struktur nahezu beliebig angelegt werden können. Das Prinzip ist dem der Objektorientierung ähnlich, in der gemäß den gegebenen Klassen und deren Struktur eine freie Anzahl von Objekten (zur Laufzeit) erzeugt werden können. Diese Ebene wird auch als ABox bezeichnet [DCTK11].

Nicht alle Sprachen zur Erstellung einer Ontologie unterstützen alle bisher beschriebenen Möglichkeiten. Besonders die frühen Sprachen haben nur Teile davon unterstützt und es entstanden immer wieder neue Sprachen, die die Mächtigkeit von Ontologien steigerten.

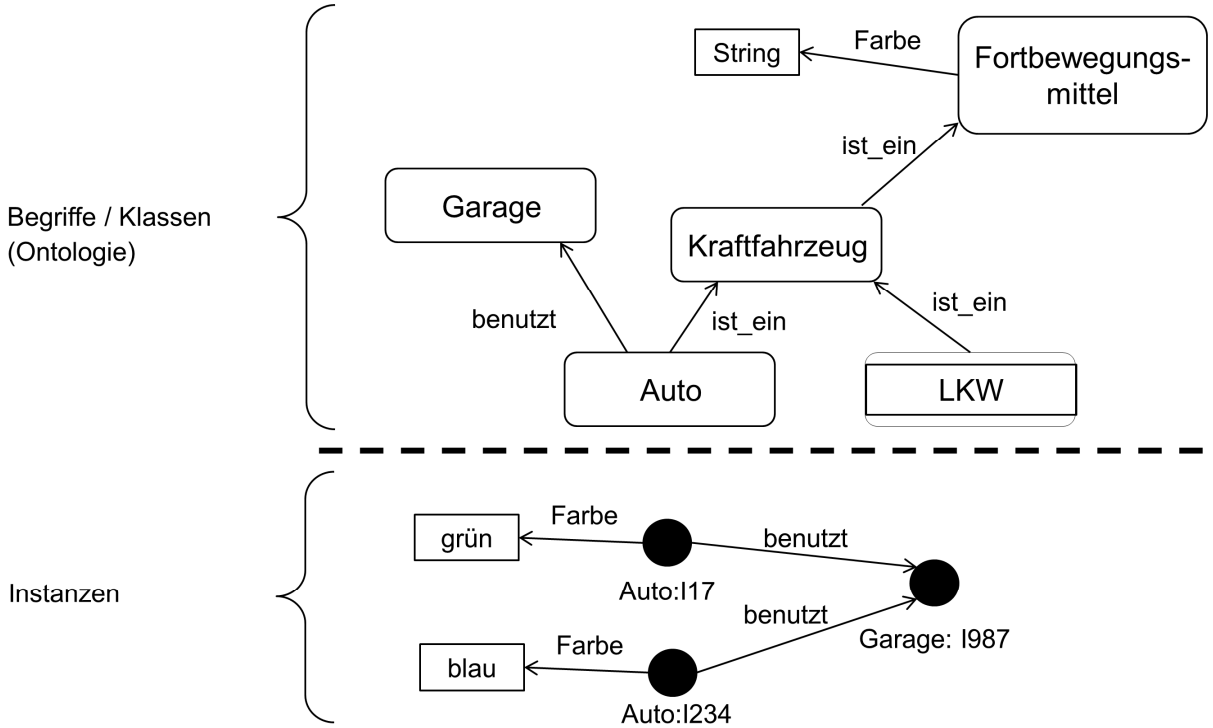


Abbildung 4.3: Die beiden Abstraktionsebenen einer Ontologie

Als eine der ersten Sprachen wurde das Resource-Description-Framework (RDF) vom W3C-Konsortium für das Semantic Web empfohlen [W3C14]. Mit dieser Sprache war es möglich, die vorgestellten Tripel abzubilden. Jedoch konnten alle möglichen Tripel und damit beliebige Aussagen, auch falsche oder sinnlose Aussagen, erstellt werden [Bröc08]. In RDF existiert nur eine einzige Abstraktionsebene für die Ontologie (nur Instanzen).

Mit dem Nachfolger RDF Schema (RDFS) [W3C14], einer Erweiterung von RDF, wird dieses Problem behoben. Mit dieser Sprache ist es möglich, Klassen von Ressourcen (Instanzen) und Relationen zu bilden. Zudem werden Begriffen Eigenschaften, hier Prädikate genannt, zugeordnet. Diese Eigenschaften sind jedoch nicht wie in der OO Bestandteil einer Klasse, sondern werden dieser wiederum über Relationen zugeordnet. Damit ist es zum Beispiel möglich, die Eigenschaften einer Klasse zu erweitern, ohne die Klasse selbst ändern zu müssen. Mit RDFS wird die zweite Abstraktionsebene (Klassen und Instanzen) in der Ontologiebeschreibung eingeführt. Jedoch ist auch hier die Ausdrucksstärke noch eingeschränkt [Bröc08].

Neben RDF und RDFS wurden mit DAML (DARPA Agent Markup Language) und OIL (Ontology Inference Layer) weitere Beschreibungssprachen für Ontologien entwickelt, die jedoch bald in einer gemeinsamen Sprache DAML+OIL und später in der Web Ontology Language (OWL) aufgingen [HPH03].

OWL [W3C12] vereint die Abbildung von Ressourcen (Instanzen) und deren Relationen aus RDF sowie deren Klassifizierung aus RDFS und ergänzt weitere Elemente. Dazu gehören Einschränkungen (z. B. Kardinalitäten oder Wertebereichsbeschränkungen) oder die Möglichkeit,

Klassen als Instanzen anderer Klassen zu definieren. Dies führt jedoch dazu, dass die beiden in Abbildung 4.3 dargestellten Abstraktionsebenen ineinander übergehen, da die klare Trennung zwischen Klassen und Instanzen nicht mehr gegeben ist. OWL ist in mehreren Sprachvarianten verfügbar: OWL Lite, OWL DL und OWL Full [Bröc08]. In dieser Reihenfolge steigen die Anzahl der Sprachkonstrukte und damit die Ausdrucksmächtigkeit, jedoch nimmt die Entscheidbarkeit auf Grund fehlender Einschränkungen (Constraints) ab.

Die überarbeitete Sprachdefinition OWL 2 [W3C12] ergänzt OWL um weiteres Vokabular, ist jedoch laut Spezifikation voll abwärtskompatibel. Hier wurden die Sprachvarianten OWL 2 EL, OWL 2 QL und OWL 2 RL eingeführt, die je nach Verwendungszweck der Ontologie eingesetzt werden können. Der Sprachstandard OWL 2 wird vom W3-Konsortium zur Verwendung empfohlen.

Um semantische Anfragen an eine Ontologie zu stellen, wird eine Abfragesprache benötigt. Ein Beispiel hierfür ist SPARQL [W3C13], das ebenfalls vom W3-Konsortium empfohlen wird und mittlerweile in der Version SPARQL 1.1 verfügbar ist. Mit Hilfe von SPARQL können unter anderem Anfragen nach den bereits vorgestellten Tripeln (Begriff – Relation – Begriff) erfolgen. Dabei ist es möglich, nach jedem Element eines Tripels zu fragen. Es können also zwei Elemente vorgegeben werden (Begriff und Relation oder Begriff und Begriff) und alle Elemente, die Teil eines Tripels sind, auf die die Beschreibung passt, werden ermittelt und zurückgegeben. Am Beispiel des Tripels *Auto – ist_ein – Kraftfahrzeug* sind die möglichen Abfragen in Abbildung 4.4 dargestellt.

| **Begriff** | **Verknüpfung** | **Begriff** | |
|---|---|---|---|
| Auto | ist_ein | **?** | → Kraftfahrzeug |
| **?** | ist_ein | Kraftfahrzeug | → Auto |
| Auto | **?** | Kraftfahrzeug | → ist_ein |

Abbildung 4.4: Mögliche Abfragen bezüglich eines Tripels

Die Suche nach einem einzelnen Tripel stellt eine sehr einfache Form dar, in der Realität werden die Abfragen schnell komplexer und es wird nach Verbindungen von mehreren Tripeln gesucht.

Das Zusammenspiel von Abfragesprache, Reasoner und Ontologie ist in Abbildung 4.5 dargestellt. Der Vergleich zu einem Datenbanksystem erleichtert das Verständnis.

Im Gegensatz zu reinen Datenbank-Systemen enthalten Ontologien die semantische Beschreibung der in ihnen enthaltenen Daten. Damit ist es möglich, den Inhalt ohne zusätzliche Informationen zu interpretieren. Dieser Umstand unterstützt das automatisierte Übersetzen von einer Ontologie in eine andere (Mapping) und das Zusammenführen von verschiedenen Ontologien

(Merging). Beim Mapping werden zwei oder mehr unabhängige Ontologien miteinander verglichen und Gemeinsamkeiten gesucht. Dabei bleibt die Unabhängigkeit der Ontologien erhalten.

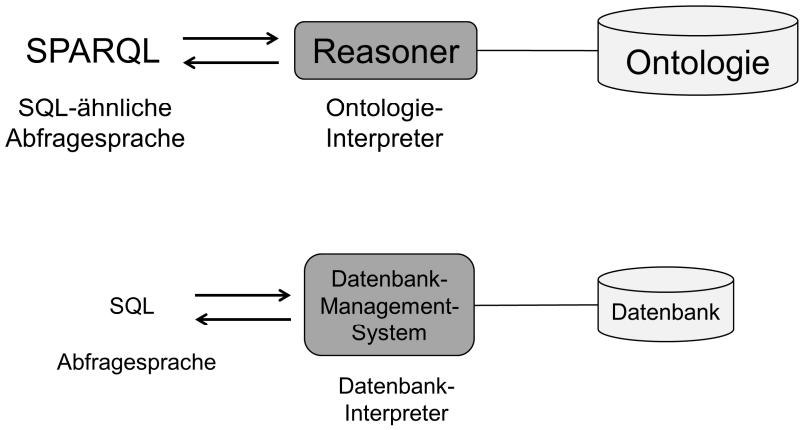


Abbildung 4.5: Ontologie vs. Datenbank

Geschieht das Mapping zur Laufzeit, so können weitere Ontologien flexibel und ohne großen Integrationsaufwand hinzugefügt werden. Dieses Konzept wird beispielsweise angewendet, wenn heterogene Datenquellen in ein Gesamtsystem integriert werden sollen [Zhan14]. Beim Merging wird, ausgehend von zwei Ontologien, eine neue Ontologie erzeugt. Diese enthält das zusammengesetzte Wissen von beiden Ausgangsontologien. Merging wird vor allem dann eingesetzt, wenn Ontologien mit unterschiedlichen Schwerpunkten zu einer umfassenden Ontologie zusammengeführt werden sollen [BoCh09]. Sowohl beim Mapping als auch beim Merging werden Ontologien auf Ähnlichkeiten bezüglich Begriffen (lexikalisch) und Relationen (semantisch) analysiert [KeLe12].

## 4.3 Erstellung von Ontologien

Zur Erstellung von Ontologien existieren verschiedene Vorgehensweisen. Beispiele hierfür sind Methontology [FGPP99], Ontology Engineering Methodology [SSS09] oder NeOn Methodology [SGF12]. Den Vorgehensweisen gemein sind verschiedene Teilaufgaben, die bearbeitet werden müssen. In [RPKC11] werden diese Teilaufgaben benannt:

- Spezifikation der Ontologie
- Wissensakquise
- Begriffsbildung
- Formalisierung
- Implementierung
- Evaluierung
- Dokumentation
- Wartung

In dieser oder ähnlicher Form sind die Schritte zur Erstellung einer Ontologie durchzuführen. Im Folgenden werden entscheidende Teilaufgaben herausgegriffen und vorgestellt.

### 4.3.1 Begriffsbildung

Einer der wichtigsten Schritte ist die Begriffsbildung (Conceptualization). Hierbei werden die Begriffe unter anderem hierarchisch angeordnet und Ober- sowie Unterklassen gebildet. Dabei gibt es verschiedene Herangehensweisen, die je nach geplantem Inhalt und Abstraktionsgrad der Ontologie mehr oder weniger gut geeignet sind. Im Folgenden werden die unterschiedlichen Ansätze vorgestellt, ausführlichere Informationen sind in [RPKC11] zu finden.

Beim Bottom-up-Vorgehen (von Unten nach Oben) wird auf der untersten Abstraktionsebene der Ontologie begonnen. Es werden sehr spezialisierte Begriffe gesammelt und durch Abstraktion wird eine Struktur gebildet. Dieses Vorgehen eignet sich vor allem für Ontologien mit niedrigem Abstraktionsgrad und einem eher kleinen und abgegrenzten Bereich, der abgebildet werden soll.

Das Gegenteil dazu ist das Top-down-Vorgehen (von Oben nach Unten). Hier wird mit sehr abstrakten Begriffen (oder Kategorien) gestartet und diese werden nach und nach spezialisiert und detailliert. Dieses Vorgehen wird angewendet, wenn eine sehr abstrakte und weit gefasste Ontologie erstellt werden soll.

Eine Kombination der beiden vorangegangenen Herangehensweisen ist das Middle-out-Vorgehen (von der Mitte nach außen). Dabei werden zunächst die wichtigsten Begriffe ermittelt und diese anschließend sowohl generalisiert als auch detailliert. Dieses Vorgehen ist geeignet zur Erstellung von Ontologien mittleren Abstraktionsgrads und führt in der Regel zu soliden Ergebnissen, da sowohl Abstraktion als auch Detaillierung berücksichtigt sind.

Je nach dem zu welchem Zweck eine Ontologie erstellt werden soll, bietet sich also eine andere Vorgehensweise an. Hinzuzufügen ist, dass Ontologien miteinander verknüpft werden können [RPKC11]. Das bedeutet, dass innerhalb einer Ontologie auf Begriffe oder Instanzen einer anderen Ontologie verwiesen bzw. eine Relation zu diesen erstellt werden kann. Ein Begriff von Ontologie A kann also beispielsweise eine Spezialisierung eines Begriffs von Ontologie B sein. Dadurch ist es möglich, mehrere kleine und spezialisierte Ontologien zu erstellen, die gemeinsam eine große und umfassende Ontologie bilden. Hierbei ist es wichtig, eine klare und sinnvolle Struktur zu finden, damit die einzelnen Teilontologien alle notwendigen Begriffe enthalten und die Gesamtontologie trotzdem beherrschbar bleibt [Bors97]. Kriterien für die vertikale Strukturierung können dabei zum Beispiel der Abstraktionsgrad der Begriffe oder der Inhalt der jeweiligen Teilontologien (allgemein/top-level bis aufgabenspezifisch) sein [RPKC11]. Oft ist es notwendig, die Ontologien zusätzlich horizontal zu strukturieren. Das bedeutet, dass nach verschiedenen Themengebieten auf demselben Abstraktionslevel getrennt wird (Bsp.: Elektrische Schaltungen – Mechanik).

### 4.3.2 Evaluierung

Ein weiterer wichtiger Schritt beim Erstellen einer Ontologie ist die Evaluierung. Die hier vorgestellten Kriterien sollten bereits in den früheren Erstellungsschritten, insbesondere bei der Begriffsbildung, berücksichtigt werden, um ein gutes Ergebnis zu erhalten. Die Kriterien sind [Vran09] entnommen. Darüber hinausgehende Kriterien und Evaluationsmethoden sind ebenfalls in [Vran09], [OCM+07] oder [LSLF14] zu finden.

- *Genauigkeit*: Die Aspekte der realen Welt müssen korrekt abgebildet sein. Das bedeutet unter anderem, dass die Axiome der Realität entsprechen müssen.
- *Anpassbarkeit*: Die Ontologie sollte für verschiedene Anwendungsfälle nutzbar sein und muss mit wenig Aufwand erweitert werden können.
- *Klarheit*: Eine Ontologie muss effektiv und objektiv sein. Zudem muss die Verständlichkeit gewährleistet sein.
- *Vollständigkeit*: Das gesamte zu beschreibende Wissensgebiet muss abgedeckt sein, das heißt alle relevanten Begriffe müssen vorhanden sein. Die Vollständigkeit kann beispielsweise mit speziellen, vor der Erstellung definierten Fragen überprüft werden.
- *Effizienz*: Die Auswertung der Ontologie innerhalb einer Software (Reasoning) muss effizient und erfolgreich sein.
- *Prägnanz*: Eine Ontologie sollte möglichst knapp gehalten werden. Das bedeutet, dass beispielsweise keine irrelevanten oder redundanten Axiome enthalten sein dürfen.
- *Konsistenz*: Die Konsistenz bezüglich verknüpften Ontologien muss gegeben sein. Ebenso dürfen keine widersprüchlichen Axiome existieren.
- *Einsatz*: Eine Ontologie sollte einfach benutzt und verbreitet werden können. Sie sollte beispielsweise keinen speziellen Beschränkungen eines Erstellungswerkzeugs unterliegen und darf nur rechtlich unbedenklichen Inhalt enthalten.

Eine unter Berücksichtigung dieser Kriterien erstellte Ontologie bildet eine gute Basis für ein Wissensbasiertes System.

## 4.4 Anwendungsbeispiele für Ontologien

Nachdem zu Beginn des Kapitels bereits festgestellt wurde, dass Ontologien in den unterschiedlichsten Bereichen Einzug erhalten, werden im Folgenden einige konkrete Anwendungsbeispiele aus dem technischen Bereich vorgestellt.

In [HEMD12] werden Ontologien benutzt, um Informationen über automatisierungstechnische Geräte aus verschiedenen Informationsquellen zusammenzuführen und zentral verfügbar zu machen. Dabei liegen die Informationen beispielsweise in Engineering-Datenbanken, Gerätedatenblättern oder Beschaffungsunterlagen vor. Die einzelnen Quellen werden in Ontologien („Serverontologien“) überführt und in einer übergeordneten Integrationsontologie zusammengefasst. So ist es zum Beispiel bei der notwendigen Ersetzung eines Gerätes möglich, alle erforderlichen Informationen zum Gerät (von Anforderungen bis zur Bezugsquelle) automatisiert zu erhalten, obwohl diese auf verschiedene Datenquellen verteilt waren. Unterstützt wird diese Informationsbereitstellung durch Hilfsontologien, die beispielsweise Ähnlichkeitsbeziehungen enthalten.

In [FKV14] werden Ontologien eingesetzt, um Modelle auf Konsistenz zu prüfen. Dazu wird ein in einer Variante von SysML vorliegendes Modell in OWL transformiert. Anschließend wird mittels gezielter Abfragen der entstandenen Ontologie, die von einem Reasoner ausgewertet werden, nach Inkonsistenzen gesucht. Jede Abfrage, die keine leere Menge zurückliefert, weist dabei auf eine Inkonsistenz hin.

Ein weiteres Beispiel für den Einsatz von Ontologien in der Automatisierungstechnik ist in [Pech14] beschrieben. Mit Hilfe eines Agentensystems und verschiedenen Ontologien wird es einem Benutzer ermöglicht, Informationsanfragen an verschiedene heterogene Datenquellen eines Automatisierungssystems zu stellen. Eine globale (allgemeingültige) Ontologie definiert die allgemeinen Begriffe und deren Verknüpfungen. An Hand dieser Begriffe kann ein Benutzer Suchanfragen erstellen. Diese werden von einem Benutzer-Agenten aufgenommen und an einen Suchanfrage-Agenten weitergegeben. Dieser analysiert die Anfrage, zerlegt sie in Teilanfragen und gibt diese an Such-Agenten weiter. Such-Agenten haben Zugriff auf lokale Ontologien, die jeweils einen speziellen Wissensbereich (z. B. Prozessführungssysteme oder ERP-Systeme) beschreiben. Mit deren Hilfe wird die Suchanfrage von der globalen in die lokale Begriffswelt übersetzt und kann so bearbeitet werden. Die verfügbaren Datenquellen eines Bereichs werden wiederum von Datenquellen-Agenten gekapselt, um das Suchsystem weitgehend unabhängig von der Realisierungsform der Datenquellen zu machen. Die Suchergebnisse werden von den Such-Agenten zurück in die globale Begriffswelt übersetzt und vom Suchanfrage-Agenten zusammengesetzt. Der Benutzer-Agent präsentiert das Ergebnis schließlich dem Benutzer. Die Kombination aus einer globalen und mehreren lokalen Ontologien ermöglicht in diesem Fall die Verknüpfung von Wissen aus verschiedenen Bereichen bzw. Begriffswelten.

In diesem Kapitel wurden Ontologien als mögliches Element einer Wissensbasis vorgestellt. Die Fähigkeit eines Reasoners, logische Schlüsse aus den Begriffen und Relationen einer Ontologie zu ziehen, macht Ontologien sehr mächtig. Deren Aufbau in Form eines semantischen Netzes mit zusätzlichen Axiomen lassen die Modellierung nahezu jeglicher Art von Wissen zu und entsprechende Abfragesprachen ermöglichen so einen breiten Einsatz. Es wurde gezeigt, dass eine

Ontologie einer Datenbank nicht unähnlich ist. Es existieren verschiedene Vorgehensweisen zur Erstellung von Ontologien, die je nach Anwendungsfall mehr oder weniger geeignet sind. Während und nach der Erstellung können diverse Kriterien herangezogen werden, um die erstellte Ontologien zu evaluieren. Schließlich wurden in diesem Kapitel an Hand von Beispielen die Einsatzmöglichkeiten von Ontologien im technischen Bereich vorgestellt. Mit Softwareagenten und Ontologien sind nun zwei mächtige Werkzeuge verfügbar, um eine flexible Prüfung von heterogenen Modellen zu realisierten. Im Folgenden wird deshalb analysiert was die Konsistenz von Modellen ausmacht und wie sie erreicht werden kann.

# 5 Bestehende Konzepte zur Konsistenzsicherung von Modellen

Inkonsistenzen in den Modellen können eine Entwicklung gefährden und/oder das entstehende Produkt unbrauchbar machen. Im Folgenden wird deshalb zunächst geklärt, was eine Inkonsistenz ist und näher auf unterschiedliche Arten von Inkonsistenzen eingegangen. Anschließend werden verschiedene Möglichkeiten zur Konsistenzsicherung, von einer manuellen Konsistenzprüfung bis hin zu einer vollautomatisierten Korrektur von Inkonsistenzen, vorgestellt. Dabei werden auch die Grenzen einer Konsistenzsicherung aufgezeigt. Schließlich werden die Anforderungen an eine automatisierte Konsistenzprüfung festgelegt. Zunächst wird jedoch aufgezeigt, was Inkonsistenzen sind.

Wie bereits in Kapitel 2 vorgestellt, werden bei der Entwicklung mechatronischer Systeme verschiedene Modelle eingesetzt. Auch in sehr vielen weiteren Entwicklungsfeldern wie beispielsweise der Architektur, der Softwaretechnik oder der Automatisierungstechnik werden Modelle verwendet, um verschiedene Aspekte des späteren Produkts bzw. Systems zu beschreiben. In der Regel wird nicht nur ein einzelnes Modell verwendet, sondern eine Menge von Modellen, die mehr oder weniger voneinander abhängig sind. Diese Abhängigkeit resultiert daraus, dass einzelne Baugruppen aus verschiedenen Sichten beschrieben werden (z. B. mechanisch und elektronisch) oder, dass dasselbe Element mit fortschreitender Entwicklung auf verschiedenen Abstraktionsebenen beschrieben ist (z. B. Klassendiagramm und Quellcode). Die Änderung eines Elements in einem Modell kann somit zu Problemen führen, wenn die Änderung nicht in die abhängigen Modelle übernommen wird und die Konsistenz nicht sichergestellt wird.

## 5.1 Klassifizierung von Inkonsistenzen

Eine Inkonsistenz tritt dann auf, wenn zwei oder mehr Elemente bezüglich desselben Sachverhalts voneinander abweichende oder gar gegensätzliche Informationen enthalten. Diese Fehler oder Widersprüche lassen sich gemäß verschiedener Kriterien klassifizieren. Im weiteren Verlauf werden einige Kriterien vorgestellt und mit Beispielen versehen, um die möglichen Folgen von Inkonsistenzen aufzuzeigen.

### 5.1.1 Modellübergreifend und modellinterne Inkonsistenzen

Inkonsistenzen können darin unterschieden werden, ob sie zwischen Elementen unterschiedlicher Modelle (modellübergreifend) oder zwischen Elementen eines einzelnen Modells (modellintern) auftreten.

Ein einfaches Beispiel für eine übergreifende Inkonsistenz ist die Änderung eines Bezeichners in Modell 1, der zur Folge haben kann, dass das betroffene Element nicht mehr mit dem korrespondierenden Element in Modell 2 in Verbindung gebracht wird, da dessen Bezeichner nicht geändert wurde. Die Auswirkungen einer weiteren Änderung können so nicht mehr ohne weiteres nachvollzogen werden und weitere Inkonsistenzen können folgen. Im schlechtesten Fall divergiert die Entwicklung an dieser Stelle.

Inkonsistenzen können jedoch auch innerhalb eines einzelnen Modells auftreten. Hier können beispielsweise nicht zulässige Verknüpfungen zwischen zwei Elementen erstellt werden, die im späteren Produkt so nicht existieren können.

### 5.1.2 Horizontale, vertikale und evolutionäre Inkonsistenzen

In der Literatur findet sich zu diesem Thema häufig die Unterscheidung von horizontaler, vertikaler und evolutionärer Konsistenz bzw. Inkonsistenz [SMSJ03, EKHG01]. Dabei versteht man unter horizontaler Konsistenz die Konsistenz mehrerer Modelle innerhalb desselben Entwicklungsschritts. Ist diese nicht gegeben, kann beispielsweise der nachfolgende Entwicklungsschritt nicht ohne Probleme durchgeführt werden, weil sich Aussagen verschiedener Modelle widersprechen.

Mit vertikaler Konsistenz wird die Konsistenz von Modellen unterschiedlicher Entwicklungsschritte bezeichnet. Sie kann auch als eine Art zeitliche Konsistenz der gesamten Entwicklung gesehen werden. Inkonsistenzen treten auf, wenn beispielsweise der Quellcode nicht korrekt vom Klassendiagramm abgeleitet wurde oder Änderungen im Quellcode durchgeführt wurden, die nicht ins Klassendiagramm übernommen sind.

Die vertikale Konsistenz steht nicht im Fokus dieser Arbeit, trotzdem sind Teile der vorgestellten Konzepte auch zu deren Prüfung einsetzbar.

Die evolutionäre Konsistenz ist die zeitliche Konsistenz eines einzelnen Modells (von einer Modellversion zur nächsten). Änderungen im Modell können zu Inkonsistenzen innerhalb des Modells führen. Ein einfaches Beispiel hierfür ist das Löschen eines Modellelements, auf das von einem anderen Modellelement aus verwiesen wird.

### 5.1.3 Formale und inhaltliche Inkonsistenzen

Inkonsistenzen werden in dieser Arbeit als formal bezeichnet, wenn kein (Fach-)Wissen zu deren Erkennen notwendig ist und diese sich nur auf die falsche Verwendung der Modellierungssprache beziehen. Eine formale Inkonsistenz bzw. ein Fehler im Modell liegt beispielsweise dann vor, wenn nicht beide Enden einer Verknüpfung zwischen Modellelementen in einem solchen enden oder ein Modellelement, das einen Bezeichner erfordert, keinen solchen besitzt. In-

konsistenzen dieser Art können oft relativ einfach durch Restriktionen in einem Editor verhindert werden.

Für das Erkennen von inhaltlichen Inkonsistenzen ist eine gewisse Menge an Wissen erforderlich. Hierzu können umfangreiche Analyseprozesse notwendig sein. Ein einfaches Beispiel ist der Abgleich der Eingangsspannung eines elektrischen Energieverbrauchers mit der Ausgangsspannung der Quelle. Aufwendiger ist die Überprüfung, ob eine elektrische Energiequelle für eine Gruppe von Verbrauchern ausreichend dimensioniert ist. Unter diese Kategorie fallen Inkonsistenzen, die sich beispielsweise auf physikalische Gesetze oder Normen usw. beziehen.

Um die Gefahr von Inkonsistenzen, die unter Umständen das gesamte Entwicklungsprojekt gefährden können, zu vermeiden, muss die Konsistenz der benutzten Modelle sichergestellt werden. Im Folgenden werden deshalb verschiedene Möglichkeiten vorgestellt, wie die Abhängigkeit von Modellen berücksichtigt werden kann.

## 5.2 Manuelle Konsistenzsicherung

Die erste Möglichkeit verschiedene, voneinander abhängige Modelle konsistent zu halten, ist die manuelle Prüfung dieser Modelle. Dabei gilt es die Modelle auf die im vorherigen Abschnitt vorgestellten Inkonsistenzen zu analysieren und gegebenenfalls Änderungen an den Modellen durchzuführen, um erkannte Inkonsistenzen aufzulösen. Dieser manuelle Prozess ist jedoch sehr zeitaufwendig und damit kostenintensiv [KRBG11a]. Zudem muss er nach jeder Anpassung der Modelle zumindest teilweise wiederholt werden [HEZ12]. Gerade bei multidisziplinären Entwicklungen ist die Menge an Wissen, die eine Person zur Prüfung aller Aspekte besitzen muss, enorm. Selbst erfahrene Entwickler stoßen dabei an ihre Grenzen [LMB09]. Dadurch steigt wiederum die Wahrscheinlichkeit von Fehlern bei der Überprüfung und die Konsistenz der Modelle ist gefährdet. Eine manuelle Konsistenzprüfung ist deshalb nur bei sehr kleinen Entwicklungsprojekten sinnvoll möglich und die Ergebnisse sind stark vom Wissen und der Erfahrung der durchführenden Person(en) abhängig.

Die Vorteile sind jedoch die Flexibilität der manuellen Durchführung und die Möglichkeit der direkten Fehlerkorrektur. Eine wichtige Voraussetzung für die Praktikabilität eines Konzepts ist die Integration in den Gesamtprozess. Je weniger am Entwicklungsprozess und an den eingesetzten Modellen und Notationen verändert werden muss, desto einfacher ist die Einführung und die Akzeptanz ist größer [SKM01]. Um Inkonsistenzen zu beseitigen, müssen diese korrigiert werden. Da dies in der Regel manuell geschieht, lässt sich dieser Vorgang sehr flexibel mit der manuellen Prüfung verbinden. So können Fehler beispielsweise direkt beim Auffinden oder gesammelt am Ende der Prüfung korrigiert werden. Eine Nachkontrolle der geänderten Modellelemente ist ebenfalls einfach zu integrieren.

Voraussetzung für jede Form der Prüfung ist das Verständnis der Modelle. Nur wenn ein Modell eindeutig und interpretierbar ist, kann eine Prüfung sinnvoll durchgeführt werden. Das bedeutet, dass der Prüfer in der Lage sein muss, mit den Modellen umzugehen, deren Aufbau zu verstehen und deren Inhalt zu interpretieren.

## 5.3 Halbautomatisierte Konsistenzsicherung

Das automatisierte Auffinden von Inkonsistenzen mit anschließender manueller Korrektur wird im Rahmen dieser Arbeit als halbautomatisierte Konsistenzsicherung bezeichnet. Dazu kommen Softwarewerkzeuge zum Einsatz, die die Modelle analysieren. Diese teilen die erkannten Inkonsistenzen dem Entwickler mit und dessen Aufgabe ist es, diese aufzulösen.

Grundvoraussetzung für den Einsatz von Softwarewerkzeugen zur Konsistenzsicherung ist das Vorliegen der Modelle in digitaler und formalisierter Form [HQRP11]. In einigen Fällen werden Algorithmen zur Konsistenzsicherung deshalb direkt in Modellierungswerkzeuge integriert.

In [Fink00] werden einige Herausforderungen vorgestellt, denen bei der Konsistenzsicherung mit Werkzeugunterstützung begegnet werden muss.

- Vage Aussagen müssen mit Hilfe formaler Modellierungssprachen ausgeschlossen werden. Alternativ können Unsicherheiten als Inkonsistenz definiert werden.
- Modelle werden bei großen Entwicklungen sehr umfangreich und die Anzahl der Modelle kann groß sein. Die Konsistenzsicherung muss also skalierbar sein.
- Eine einzelne Inkonsistenz kann ein gesamtes Modell unbrauchbar machen oder andere Inkonsistenzen verdecken.
- Modelle können sich, vor allem wenn sie verteilt entwickelt werden, auf einem unterschiedlichen Entwicklungsstand befinden. Eine Prüfung ist damit nicht zu jedem beliebigen Zeitpunkt sinnvoll.

Ansätze, die sich mit der Konsistenzsicherung beschäftigen, gibt es im Bereich der Softwaretechnik schon seit vielen Jahren [FGH+94]. Die Softwaretechnik ist durch die fast ausschließliche Verwendung von formalen Modellen für diesen Zweck begünstigt. Durch die Verbreitung von computergestützten Modellierungswerkzeugen und damit der Einführung von formaleren Modellierungssprachen (z. B. XML) in anderen Bereichen, werden Konzepte zur automatisierten Konsistenzsicherung auch dort verfolgt. Einfachstes Beispiel ist die Rechtschreib- oder Grammatikprüfung in Textverarbeitungsprogrammen, aber auch in der Architektur existieren Ansätze, Modelle auf Konsistenz zu prüfen [Lee10].

In [SKM01] wird ein Verfahren zur Prüfung von UML-Diagrammen (Modelle) vorgestellt. Zustands- und Sequenzdiagramme (Kollaborationsdiagramme) beschreiben das Verhalten eines

Systems auf unterschiedliche Art und in der Regel zu verschiedenen Entwurfszeitpunkten. Sie müssen jedoch konsistent sein, um eine erfolgreiche Implementierung zu ermöglichen. Der Ansatz sieht vor, die Sequenzdiagramme in Automaten zu übersetzten und mit Hilfe eines Modelcheckers (SPIN) gegenüber den Zustandsdiagrammen zu überprüfen.

Ein anderer Ansatz zur Prüfung von UML-Diagrammen findet sich in [SMSJ03]. Die Diagramme werden in Description Logic (DL) übersetzt und mit Hilfe von Regeln überprüft. Aufgetretene Inkonsistenzen können laut den Autoren bereits automatisiert aufgelöst werden, jedoch war die Übersetzung der Diagramme in DL beim damaligen Stadium noch manuell durchzuführen. Gleiches gilt für die Auflösung der Inkonsistenzen.

Ein Werkzeug zur Prüfung von mechatronischen Modellen wird in [HEZ10] vorgestellt. Die Aktionen eines Entwicklers im Modellierungswerkzeug werden erkannt und das Modell gezielt auf Inkonsistenzen geprüft. Der Vorteil dabei ist, dass nicht das gesamte Modell geprüft werden muss, sondern nur die Auswirkungen der aktuell erkannten Änderungen verfolgt werden müssen. Das ergibt einen enormen Geschwindigkeitsvorteil und ermöglicht die Prüfung parallel zur Arbeit der Entwickler. Die Prüfung selbst wird mit Hilfe von Konsistenzregeln durchgeführt, die so formuliert sind, dass alle erfüllt sein müssen, um die Konsistenz der Modelle zu bestätigen.

Die vorgestellten Ansätze beschränken sich darauf, dem Entwickler Inkonsistenzen aufzuzeigen. Deren Auflösung bleibt die Aufgabe des Entwicklers. Allen Beiträgen gemein ist jedoch die immense Zeitersparnis durch die automatisierte Prüfung mit der der Entwickler bei seiner Tätigkeit unterstützt wird.

## 5.4 Vollautomatisierte Konsistenzsicherung

Bei der vollautomatisierten Konsistenzsicherung, wird dem Entwickler nicht nur die Prüfung von Modellen von einem Werkzeug abgenommen sondern auch die Auflösung von Inkonsistenzen. Ein etwas anderer Ansatz, die Modell-zu-Modell-Transformation versucht Inkonsistenzen zu vermeiden, indem Modelle automatisiert von anderen Modellen abgeleitet werden. In Folgenden werden Beispiele für beide Ansätze vorgestellt.

### 5.4.1 Modell-zu-Modell-Transformation

Die Modell-zu-Modell-Transformation transformiert ein Modell in ein anderes Modell [Maur13]. Die Modelle besitzen häufig unterschiedliche Modellierungssprachen, sie können jedoch auch in derselben Sprache beschrieben sein. Ebenso kann der Inhalt der Modelle vor und nach der Transformation derselbe sein oder es werden Informationen ergänzt bzw. reduziert. Transformationen können uni- oder bidirektional sein.

Für die Transformation wird zunächst das Ausgangsmodell (Quellmodell) analysiert und nach bestimmten Mustern untersucht. Diese Muster sind mit Transformationsvorschriften verknüpft,

welche beim Auffinden eines entsprechenden Musters angewendet werden. Eine sogenannte Plattform enthält Sprachkonstrukte, Gebrauchsmuster und vordefinierte Instanzen, die im Zielmodell eingesetzt werden können. Die Transformationsvorschriften definieren welche dieser Elemente im Zielmodell wo eingefügt werden. Das Prinzip der Modell-zu-Modell-Transformation ist in Abbildung 5.1 dargestellt.

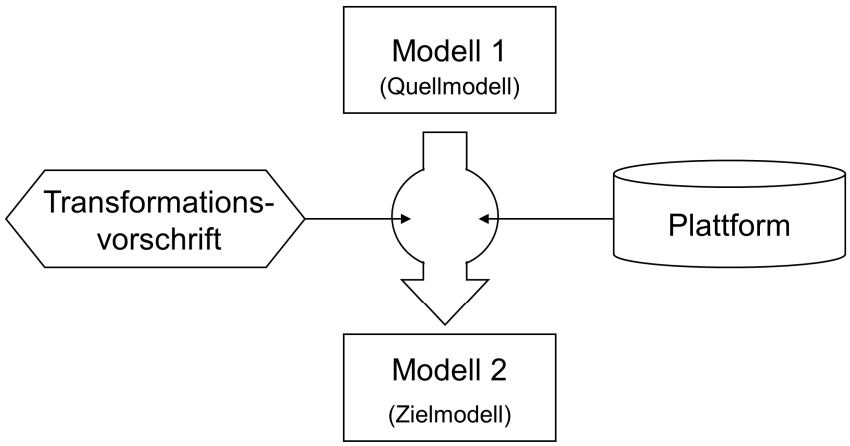


Abbildung 5.1: Schematische Darstellung einer Modell-zu-Modell-Transformation

In [Maur13] wird eine solche Transformation vorgestellt, die bei der Entwicklung von Automatisierungssystemen eingesetzt werden kann. Modelle, die einen technischen Prozess in einer domänenspezifischen Sprache beschreiben, werden unter Einsatz einer komponentenbasierten Plattform automatisiert in Quellcode der Automatisierungssoftware und Schaltpläne für die Hardware übersetzt. So wird einerseits der Entwicklungsaufwand reduziert, andererseits werden Fehlerquellen durch die Automatisierung minimiert. Sofern die Transformationsvorschriften korrekt und vollständig sind und das Zielmodell nicht nachträglich manuell verändert wird, existieren keine Inkonsistenzen zwischen Quell- und Zielmodell.

Ein Beispiel für die Transformation von SysML-Modellen (erweitert für die Automatisierungstechnik) in Quellcode für Steuerungsprogramme wird in [VSFL14] vorgestellt. Hierbei werden zusätzliche Verknüpfungen zwischen Modell und Steuerungsprogramm geschaffen und der aktuelle Zustand der Steuerung im Modell dargestellt.

Ein anderes Beispiel, das über die reine Transformation hinausgeht, findet sich in [RSRV14]. Dort wird eine Modellsynthese vorgestellt, bei der zusätzlich zur reinen Transformation verschiedene Parameter für Komponenten berechnet werden.

Die Modell-zu-Modell-Transformation wird in der Regel angewendet, um Modelle eines Entwicklungsschritts in Modelle eines anderen Entwicklungsschritts zu transformieren. Sie zielt deshalb auf die vertikale Konsistenzsicherung ab.

### 5.4.2 Automatisierte Konsistenzsicherung

Die automatisierte modellübergreifende Konsistenzsicherung geht von zwei oder mehr von (menschlichen) Entwicklern erstellten Modellen aus. Beide Modelle können von den Entwicklern unabhängig verändert werden und die Konsistenz muss sichergestellt werden. Dies geschieht automatisiert, indem Änderungen in einem Modell ohne Zutun der Entwickler auf andere Modelle übertragen werden. Inkonsistenzen werden also erkannt und automatisiert behoben.

In [GSG+09] wird ein Ansatz vorgestellt, der es ermöglicht, mit Hilfe von Triple Graph Grammatiken (TGG) Änderungen in einem Modell auf andere Modelle zu übertragen. TGGs bestehen aus Produktionsregeln, die den Quellgraphen und den Zielgraphen berücksichtigen [Schü95]. Wird ein bestimmtes Muster im Quellgraphen gefunden, kann eine Produktionsregel angewendet und somit der Zielgraph entsprechend einem anderen Muster erweitert werden. Der Einsatz von TGGs ermöglicht einerseits die initiale Transformation eines Modells in ein anderes, andererseits die Übertragung von Änderungen von einem Modell zum anderen. Die Autoren von [GSG+09] transformieren mit TGGs ein Modell der mechatronischen Prinziplösung (früher Entwurf) in ein disziplinspezifisches Modell des späten Entwurfs. In beiden Modellen können anschließend Modellelemente hinzugefügt werden, die automatisiert in das jeweils andere Modell übernommen werden. Mit diesem Ansatz können strukturelle evolutionäre Inkonsistenzen vermieden werden.

In [FBRG06] wird berichtet, wie zwei UML-Modelle unter Anwendung von (Zustands-) Automaten konsistent gehalten werden können. Es wird davon ausgegangen, dass jede Änderung in einem Modell einer Änderung von dessen Zustand entspricht. Der entsprechende Zustandsübergang bzw. die Zustandsübergänge werden ermittelt und auf das andere Modell übertragen. Dort werden dieselben Zustandsübergänge ausgeführt und die entsprechenden Änderungen im Modell durchgeführt. Im vorgestellten Ansatz werden die Zustandsautomaten noch manuell erstellt, die Nutzung von Query-View-Transformations (QVT), die den TGGs ähnlich sind, werden jedoch angekündigt.

In beiden vorgestellten Ansätzen können Modelle bidirektional auf Inkonsistenzen geprüft und diese aufgelöst werden. Eine Inkonsistenz innerhalb eines Modells wird jedoch nicht erkannt und der Fokus liegt auf der Struktur der Modelle.

Der in [KRBG11a] vorgestellte Ansatz prüft ein einzelnes CAD-Modell der Mechanik auf Konsistenz und bietet dem Entwickler Verbesserungsvorschläge an. Die Prüfung erfolgt mit Hilfe eines Agentensystems, das Änderungen des Entwicklers im Modell selbstständig erkennt und die Konstruktion auf die Einhaltung von Normen und technologischen Randbedingungen analysiert. Werden Probleme erkannt, so wird durch Variation verschiedener Parameter automatisiert eine Lösung ermittelt und dem Entwickler vorgeschlagen. Mit dessen Einverständnis wird diese im Modell umgesetzt und die Konsistenz ist sichergestellt. Änderungen der Struktur sind bei der Lösungsermittlung jedoch nicht möglich.

Den vorgestellten Ansätzen ist gemeinsam, dass Inkonsistenzen nur in gewissem Rahmen vermieden oder automatisiert aufgelöst werden können. Entweder dürfen, wie bei der Modell-zu-Modell-Transformation, Quell- und Zielmodell nach der Transformation nicht mehr verändert werden oder es müssen sämtliche Entwurfsmuster (Elementkombinationen) hinterlegt sein, wie es beispielsweise bei den TGGs der Fall ist. Zudem konzentrieren sich die Ansätze entweder auf die Struktur der Modelle oder auf deren Inhalt. Eine umfassende Prüfung mit automatisierter Lösungsermittlung gibt es derzeit noch nicht.

## 5.5 Anforderungen an die Konsistenzsicherung

Die Aufgabe der Konsistenzsicherung ist es, Inkonsistenzen aufzudecken und zu beseitigen. Dabei ist die Aufdeckung von Inkonsistenzen zwar eine aufwendige, jedoch wenig kreative Tätigkeit. Im Gegensatz dazu ist deren Beseitigung in der Regel eine Aufgabe, die Kreativität erfordert. Diese Kreativität kann nur realisiert werden, wenn eine sehr große Anzahl an Entwurfsmustern hinterlegt wird, die ausgewählt und eingesetzt werden können. Dieser Aufwand ist so groß, dass es effektiver erscheint, einem menschlichen Benutzer die Inkonsistenz mit allen Details aufzuzeigen und diesen damit bei deren Auflösung zu unterstützen. Die automatisierte Auflösung von Inkonsistenzen ist zwar wünschenswert, aber nicht im Fokus dieser Arbeit. Einem Benutzer (Entwickler) sollen jedoch alle verfügbaren Informationen bereitgestellt werden, damit dieser die Inkonsistenz mit dem kleinstmöglichen Aufwand beseitigen kann. Im weiteren Verlauf wird die Konsistenzsicherung deshalb auf die Prüfung der Modelle reduziert.

Aus den vorgestellten Beispielen lassen sich zudem die folgenden Anforderungen an die Konsistenzprüfung ableiten:

**Anforderung 1:**

Der Entwicklungsprozess und die zugehörigen Abläufe sollen so wenig wie möglich beeinflusst werden. Das bedeutet, dass die Prüfung in die bestehenden Prozesse eingebunden werden muss.

**Anforderung 2:**

Die Prüfung auf Inkonsistenzen muss zudem zeitlich flexibel erfolgen können. Es muss also möglich sein, jederzeit eine Prüfung zu starten, wenn eine solche von den Entwicklern gewünscht ist. Wünschenswert ist auch die Möglichkeit, die Prüfung permanent im Hintergrund auszuführen, um Inkonsistenzen den Entwicklern direkt anzuzeigen.

**Anforderung 3:**

Eine zeitlich flexible Prüfung muss auch inhaltlich flexibel sein. Es sind nicht zu jedem Zeitpunkt alle Modelle im gleichen Detaillierungsgrad vorhanden und es werden im Laufe der Entwicklung Modelle hinzugefügt. Die Prüfung kann und muss also immer auf die verfügbaren Informationen bzw. Modelle angewendet werden können.

**Anforderung 4:**

Modelle sind, vor allem bei disziplinübergreifenden Entwicklungen wie der Mechatronik, in der Regel nicht einheitlich. Die Prüfung muss also auf heterogene Modelle angewendet werden können. Die Integration eines neuen Modells muss zur Laufzeit möglich sein.

**Anforderung 5:**

Die Prüfung muss Inkonsistenzen erkennen. Diese Inkonsistenzen können innerhalb eines einzelnen Modells auftreten oder zwischen verschiedenen Modellen vorhanden sein.

**Anforderung 6:**

Dem Benutzer müssen beim Auffinden einer Inkonsistenz alle zu deren Auflösung verfügbaren Informationen bereitgestellt werden, damit dieser so gut wie möglich unterstützt wird. Zu diesen Informationen gehören die genaue Stelle im Modell, an der die Inkonsistenz aufgetreten ist, und die Begründung, warum die Inkonsistenz erkannt wurde. Je nach Art der Inkonsistenz ist es wünschenswert, einen Lösungsvorschlag oder zumindest einen Hinweis zu deren Auflösung zu erhalten.

In diesem Kapitel wurde erläutert, was eine Inkonsistenz ist und es wurden verschiedene Kriterien zu deren Klassifizierung genannt. Die möglichen Konzepte zur Konsistenzsicherung von Modellen mit manueller, halbautomatisierter und vollautomatisierter Durchführung wurden vorgestellt. Diese besitzen unterschiedlich viel manuellen Aufwand für den Entwickler, jedoch ist eine vollautomatisierte Konsistenzsicherung in vielen Fällen auf Grund des Umfangs des dazu benötigten Wissens (noch) nicht realisierbar. Deshalb ist es sinnvoll, lediglich die Prüfung der Modelle zu automatisieren. Diese ist eine wenig kreative Tätigkeit und kostet die Entwickler viel Zeit. Die viel Kreativität benötigende Behebung von Inkonsistenzen bleibt somit den Entwicklern überlassen. Diese sollen dabei jedoch so gut wie möglich unterstützt werden. Am Ende des Kapitels wurde dies in den Anforderungen an eine solche automatisierte Prüfung festgelegt. Im folgenden Kapitel wird das Konzept für die Prüfung vorgestellt. Ebenso werden die notwendigen Voraussetzungen und Vorarbeiten präsentiert.

# 6 Konzept zur agentenbasierten Konsistenzprüfung von heterogenen Modellen

Die Prüfung von heterogenen Modellen bedarf eines anderen Vorgehens als dies bei homogenen Modellen der Fall ist. Im Folgenden wird, ausgehend von einem menschlichen Prüfer, der prinzipielle Ablauf im Umgang mit heterogenen Modellen vorgestellt. Dazu werden die verschiedenen Abstraktionsebenen sowie die grundsätzlichen Zusammenhänge zwischen den Modellen präsentiert. Anschließend wird ein Überblick über das Konzept zur agentenbasierten Konsistenzprüfung heterogener Modelle gegeben. Es wird beschrieben, welche Ideen hinter dem Konzept stehen und wie Agenten und Ontologien eingesetzt werden sollen. Zudem wird der Ablauf einer Prüfung beschrieben. Schließlich werden die Voraussetzungen aufgezeigt, die notwendig sind, um das Konzept auf heterogene Modelle anwenden zu können. Eine durch das Softwaresystem bedingte Voraussetzung ist das Vorliegen der zu prüfenden Modelle in einer für das Softwaresystem verarbeitbaren Form. Die unterschiedlichen Modelle müssen deshalb formalisiert und abstrahiert werden, um sie auslesen und interpretieren zu können. Es werden dazu zunächst die verschiedenen Abstraktionsebenen aufgezeigt, die von der Realität bis hin zu einem zu entwickelnden Meta-Meta-Modell reichen. Anschließend werden die zur Abstraktion notwendigen Schritte am Beispiel der Wirkstruktur vorgestellt. An Hand von weiteren Modellen wird die Anwendbarkeit des entstandenen Meta-Meta-Modells aufgezeigt. Schließlich werden die Grenzen der Abstraktion erläutert.

## 6.1 Prinzipielles Vorgehen bei der Prüfung von Modellen

Bei der Prüfung von Modellen werden die Aussagen, die in den einzelnen Modellen gemacht werden bzw. modelliert sind, auf ihre Konsistenz zueinander geprüft. Aussagen, die im Widerspruch zueinander stehen, bedeuten dabei eine Inkonsistenz. Bei homogenen Modellen, die alle in der gleichen Modellierungssprache beschrieben sind und vom gleichen Modelltyp sind, gestaltet sich der Vergleich der Aussagen relativ einfach. Die Aussagen wurden alle in derselben Sprache formuliert und können direkt gegenübergestellt werden.

Ein menschlicher Prüfer wird die Aussagen der Modelle mit Hilfe des ihm eigenen Wissens interpretieren. Dazu wird er sie bewusst oder unbewusst in seine Vorstellung, die wiederum von seinem Wissen abhängig ist, abstrahieren. Auf dieser abstrakten Ebene wird er anschließend die Aussagen mit den in seinem Wissen verankerten Regeln und Gesetzen vergleichen. Ein dem menschlichen Prüfer unbekanntes Modell eines ihm nicht geläufigen Modelltyps, modelliert in einer ihm unbekannten Modellierungssprache, kann er dagegen nicht ohne weiteres abstrahieren und interpretieren. Dazu ist es erforderlich, dass er unter anderem die Bedeutung der einzelnen Symbole der verwendeten Modellierungssprache kennt. Mit Hilfe einer Beschreibung bzw. In-

terpretationsanleitung, kann er sich dieses Wissen jedoch aneignen und aus dem ihm zuvor unbekannten Modelltyp wird ein ihm bekannter Modelltyp.

Ein analoges Vorgehen ist in dem vorliegenden Konzept vorgesehen. Das Wissen bzw. die bei einer Prüfung anzuwendenden Regeln sind abstrakt und damit modell- und modelltypunabhängig formuliert. Modelle, die geprüft werden sollen, müssen damit zunächst auf dieselbe Ebene abstrahiert werden. Abbildung 6.1 zeigt dieses Vorgehen.

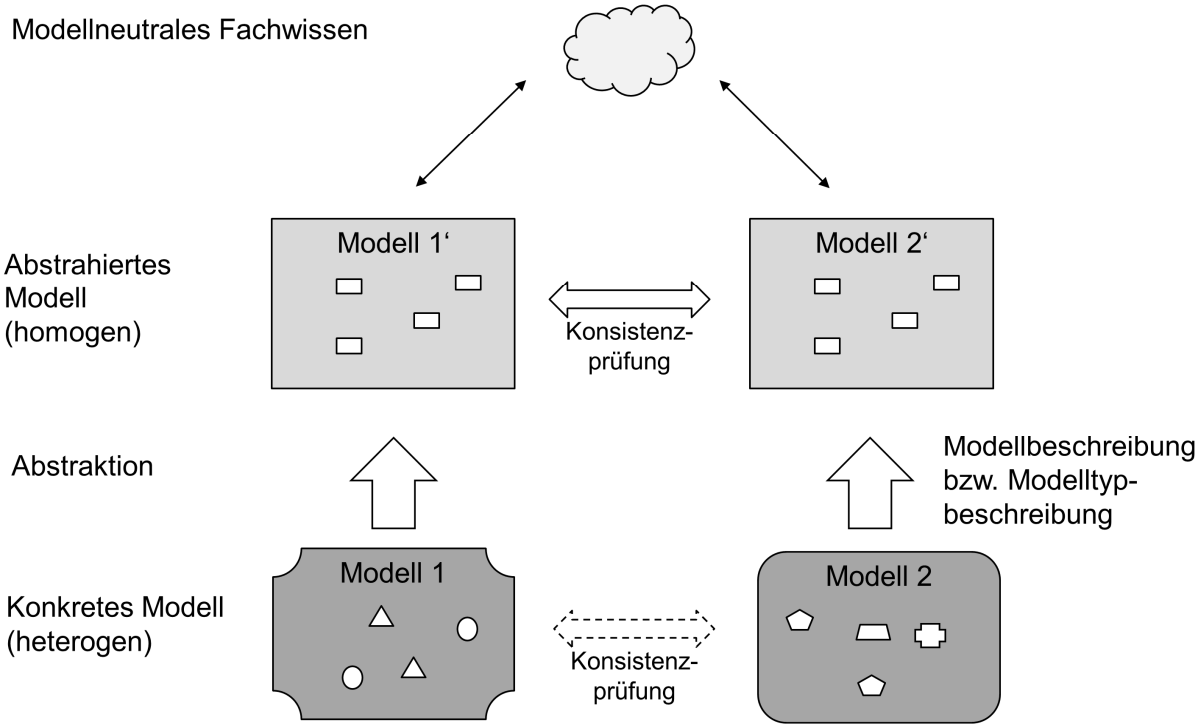


Abbildung 6.1: Prinzipielles Vorgehen für die Interpretation von heterogenen Modellen

Die Aussagen zweier unterschiedlicher Modelle bzw. Modelltypen können nicht direkt verglichen werden, insbesondere wenn in beiden Modellen unterschiedliche Symbole (Modellelemente) verwendet werden. Sie müssen zuerst auf eine allgemeine, modelltypunabhängige Ebene abstrahiert werden. Diese muss so mächtig sein, dass die Inhalte aller zu prüfender Modelle darin abgebildet werden können. Sind die Modelle einmal darin abgebildet, ist ihre Darstellung nicht mehr heterogen sondern homogen, da alle modelspezifischen Elemente durch die Abstraktion entfernt wurden. Diese Ebene soll deshalb im weiteren Verlauf als globale Ebene oder globale Begriffswelt bezeichnet werden.

Im Gegensatz dazu wird die Ebene, in der sich die einzelnen Modelle befinden, als lokale Ebene bezeichnet. Jedes Modell bzw. jeder Modelltyp kann prinzipiell einen anderen Aufbau und andere Modellelemente besitzen. Außerdem können die Modellelemente und allgemein die modellierten Objekte mit unterschiedlichen Begriffen bezeichnet werden. Jeder Begriff kann somit einer eigenen, der sogenannten lokalen Begriffswelt entnommen sein. Ein Überblick über die Verwendung der lokalen sowie der globalen Begriffswelt ist in Abbildung 6.2 dargestellt.

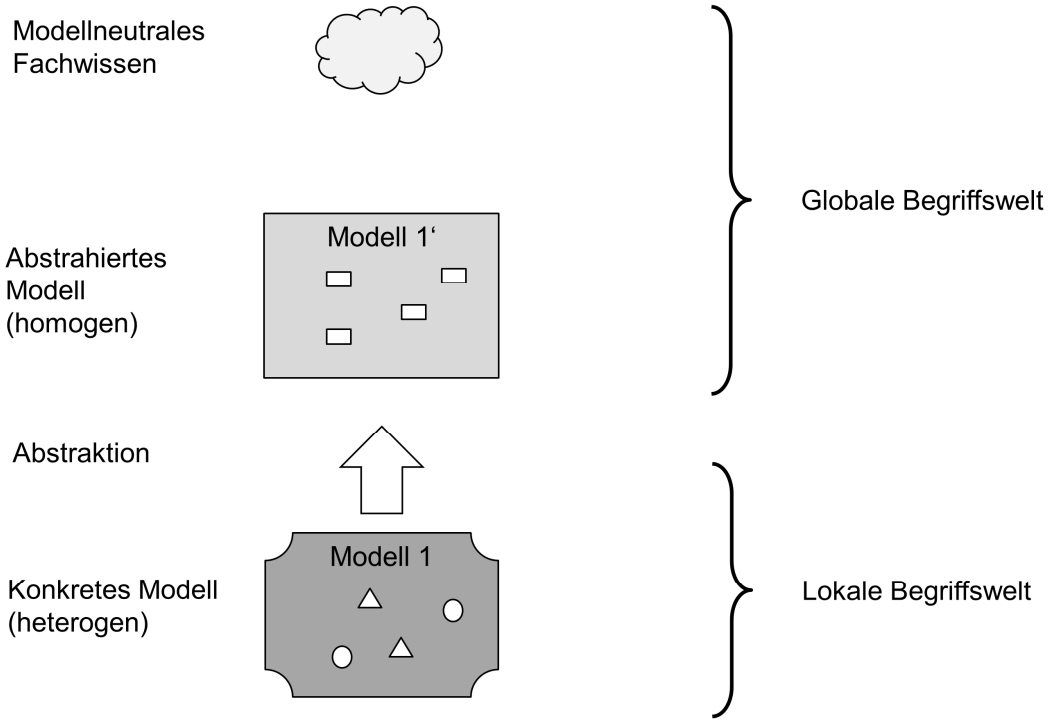


Abbildung 6.2: Verwendung von lokaler und globaler Begriffswelt

Für die Abstraktion der Modelle von der lokalen in die globale Ebene sind unter anderem entsprechende Modellbeschreibungen notwendig. Diese Modellbeschreibungen sind spezifisch für jeden Modelltyp und enthalten alle Informationen, um die Modelle von der lokalen, modellspezifischen Ebene in die globale Ebene zu transferieren. Diese Informationen beinhalten beispielsweise eine Beschreibung der in einem Modelltyp verwendbaren Symbole, deren Verknüpfungsmöglichkeiten und deren Bedeutung, die als eine Art Übersetzung zwischen lokaler und globaler Begriffswelt dient. Der detaillierte Inhalt und der Aufbau einer solchen Modellbeschreibung wird in Kapitel 8.2 vorgestellt werden.

## 6.2 Überblick über das Konzept

Das vorliegende Konzept beschreibt, wie heterogene Modelle rechnergestützt geprüft werden können. Dazu wird die im vorherigen Unterkapitel vorgestellte Idee genutzt und die Modelle werden auf eine modellneutrale Ebene abstrahiert, um dort geprüft zu werden. Die Abstraktion sowie die Prüfung werden dabei von einem Softwaresystem durchgeführt. Um dies zu ermöglichen, müssen einerseits die Modelle und andererseits das für die Prüfung benötigte Wissen in einer für ein Rechnersystem verarbeitbaren Form vorliegen. Eine Form, die sich zudem einfach erweitern lässt, ist eine Ontologie.

Das modellneutrale Fachwissen der globalen Begriffswelt wird deshalb in einer Ontologie abgebildet. Sie enthält Fakten, aus denen wiederum Regeln für die Überprüfung abgeleitet werden. Zudem enthält sie explizit formulierte Regeln. Da diese Ontologie das allgemeine, modellüber-

greifende Fachwissen enthält und die globale Begriffswelt definiert, wird sie im weiteren Verlauf als globale Ontologie bezeichnet.

Die Modelltypbeschreibungen werden ebenfalls in Form von Ontologien abgebildet. So wird jedem Modell bzw. Modelltyp eine Ontologie zugeordnet, die die Transformation eines Modells von der lokalen in die globale Begriffswelt ermöglicht. Diese Ontologie ist die Verbindung zwischen Modell und globaler Ontologie oder dem darin repräsentierten Fachwissen. Da diese Ontologie die lokale Begriffswelt repräsentiert, wird sie als lokale Ontologie bezeichnet. Abbildung 6.3 zeigt die beiden Ontologietypen im Zusammenhang.

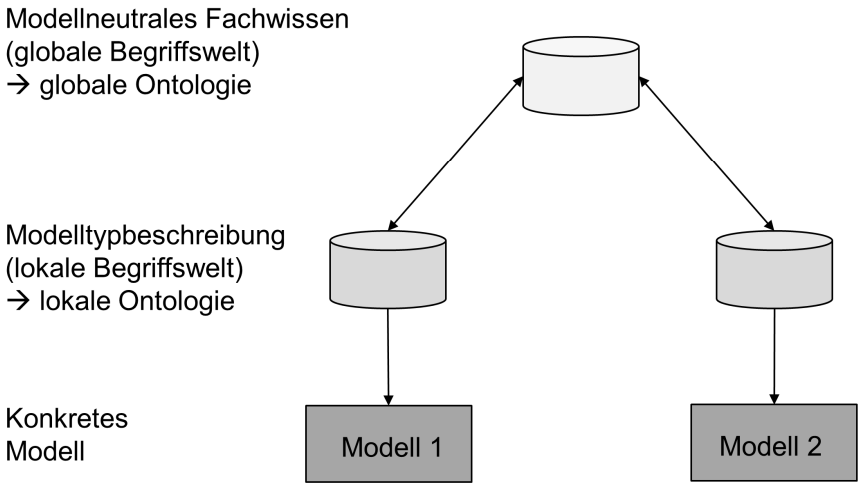


Abbildung 6.3: Repräsentation von Fachwissen und Modelltypbeschreibung in Form von Ontologien

Das Prinzip zur Verknüpfung heterogener Datenquellen mittels lokaler und globaler Ontologien wird in ähnlicher Form in [Pech14] angewendet.

Der Inhalt bzw. die Informationen eines Modells müssen im System abgebildet werden. Dazu dient ein abstrakter, modellübergreifender Informationscontainer. Er ist eine Art abstraktes Übermodell über alle Modelltypen und ist in der Lage, alle Modelle abzubilden.

Die Prüfung der Modelle muss flexibel erfolgen, sie soll selbstständig ablaufen und die Schnittstellen zu den heterogenen Modellen müssen gekapselt werden. Zudem muss das System zur Prüfung einfach erweiterbar sein. Zentrale Ansätze stoßen hierbei bezüglich Komplexität an die Grenzen. Darum bietet sich der Einsatz von Softwareagenten an. Sie repräsentieren einerseits die einzelnen Modelle, andererseits vertreten sie das globale Wissen sowie die daraus abgeleiteten Überprüfungsregeln und sie koordinieren die Prüfung. Abbildung 6.4 zeigt die im Konzept eingesetzten Agenten im Überblick.

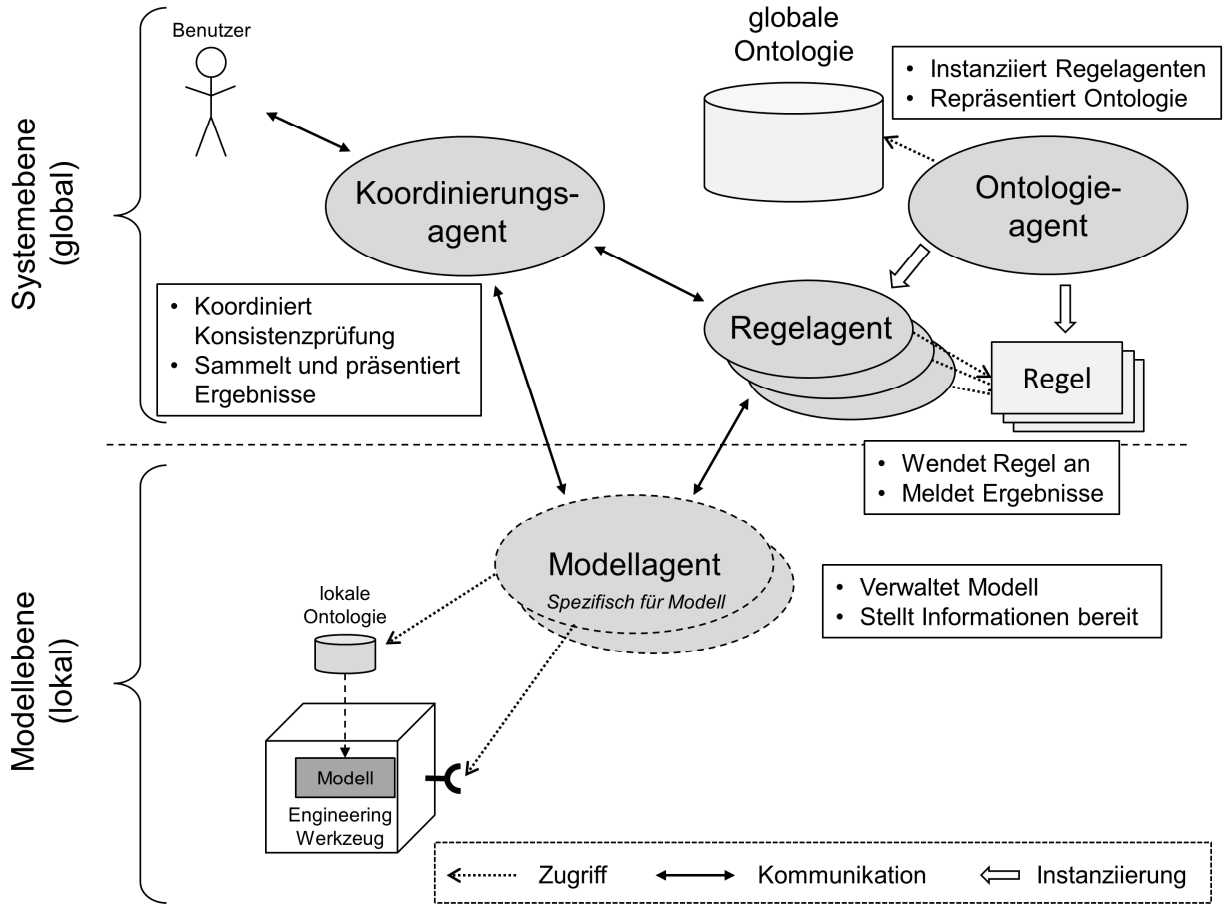


Abbildung 6.4: Überblick über die verschiedenen Agenten des Konzepts

Der Koordinierungsagent koordiniert die Prüfung der Modelle, stößt diese an und bestimmt das Ende der Prüfung. Die einzelnen Ergebnisse der Prüfungen laufen bei ihm zusammen und er bereitet diese für den Benutzer auf. Er dient somit als Schnittstelle zum Benutzer.

Der Ontologieagent repräsentiert die globale Ontologie und damit das Fachwissen. Er leitet aus diesem Wissen Überprüfungsregeln ab und instanziiert zugehörige Regelagenten. Die Prüfung selbst ist jedoch nicht Aufgabe dieses Agenten.

Die instanziierten Regelagenten repräsentieren die ihnen zugeordnete Regel und wenden diese im Auftrag des Koordinierungsagenten auf die Modelle an bzw. prüfen die Modelle auf Einhaltung der jeweiligen Regel. Die notwendigen Informationen aus den Modellen erhalten sie von den Modellagenten. Die Ergebnisse der Regelanwendung werden schließlich dem Koordinierungsagenten gemeldet.

Modellagenten repräsentieren jeweils ein Modell und stellen die darin enthaltenen Informationen bereit. Sie haben Zugriff auf die Modellbeschreibung in Form der lokalen Ontologie und transformieren das Modell mit deren Hilfe in die globale Ebene. Dort kapseln sie das Modell und stellen eine einheitliche, vom Modelltyp unabhängige Schnittstelle für die Regelagenten bereit. Lediglich die Schnittstelle zum Engineeringwerkzeug kann spezifisch realisiert sein. Die

Modellagenten sind damit das Bindeglied zwischen den heterogenen Modellen und dem globalen (homogenen) Fachwissen.

Bei einer Prüfung wird der Prüfungsprozess vom Benutzer über den Koordinierungsagenten angestoßen. Dieser beauftragt wiederum die im System vorhandenen Regelagenten ihre jeweiligen Regeln auf die vorhandenen Modelle anzuwenden und zu überprüfen. Die Regelagenten kommunizieren dazu mit den Modellagenten, um die erforderlichen Informationen aus den Modellen zu erlangen. Die Ergebnisse der Regelanwendung sowie alle Informationen, die zum jeweiligen Ergebnis geführt haben, werden von den Regelagenten an den Koordinierungsagenten übermittelt. Dieser bereitet die Ergebnisse für den Benutzer auf und präsentiert ihm diese. Der Benutzer kann so genau nachvollziehen, welche Information aus welchem Modell warum zu einer nichterfüllten Regel geführt hat und das Problem beheben.

## 6.3 Voraussetzungen im Umgang mit heterogenen Modellen

Um heterogene Modelle nach dem vorgestellten Konzept zu prüfen, müssen verschiedene Voraussetzungen gegeben sein, die in den folgenden Abschnitten vorgestellt werden. Ein heterogenes Modell ist zunächst mit einem dem System unbekannten Modell gleichzusetzen. Das System ist also nicht speziell an dieses Modell angepasst. Damit das Modell im System abgebildet und verarbeitet werden kann, müssen dessen Informationen innerhalb des Systems in einem neutralen Informationscontainer bereitgestellt werden können. Es wird ein gemeinsames, übergeordnetes Modell benötigt, in dem die in einem beliebigen konkreten Modell enthaltenen Informationen abgebildet werden können. Dieses abstrakte, übergeordnete Modell wird im weiteren Verlauf als Meta-Meta-Modell bezeichnet und in den folgenden Unterkapiteln entwickelt.

Damit die abgebildeten Informationen eines Modells interpretiert werden können, muss deren Bedeutung ebenfalls im System vorhanden sein. Dazu muss zu jedem Modell eine entsprechende Modellbeschreibung vorliegen (vgl. [Arnd13]). Für diese Beschreibung ist es jedoch ausreichend, wenn sie nicht das konkrete Modell, sondern das Meta-Modell bzw. die Beschreibungssprache enthält. Sie beschreibt also für den jeweils vorliegenden Modelltyp (Meta-Modell), wie das konkrete Modell zu interpretieren ist. Diese Modellbeschreibung wird in Kapitel 8.2 präsentiert.

Mit dem Inhalt eines konkreten Modells und der Beschreibung, wie dieser Inhalt zu interpretieren ist, kann eine Prüfung erst durchgeführt werden, wenn das Wissen über Fakten und Zusammenhänge vorhanden ist. Nur dann kann mit Hilfe von Regeln entschieden werden, ob der Inhalt der Modelle mit diesem Wissen konform und damit, ob die Modelle konsistent sind. Dieses Wissen muss unabhängig von den Modellen sein, damit es auf heterogene Modelle angewendet werden kann. Wie dieses Wissen aufgebaut ist und wie die Regeln daraus abgeleitet werden, wird in Kapitel 8 beschrieben.

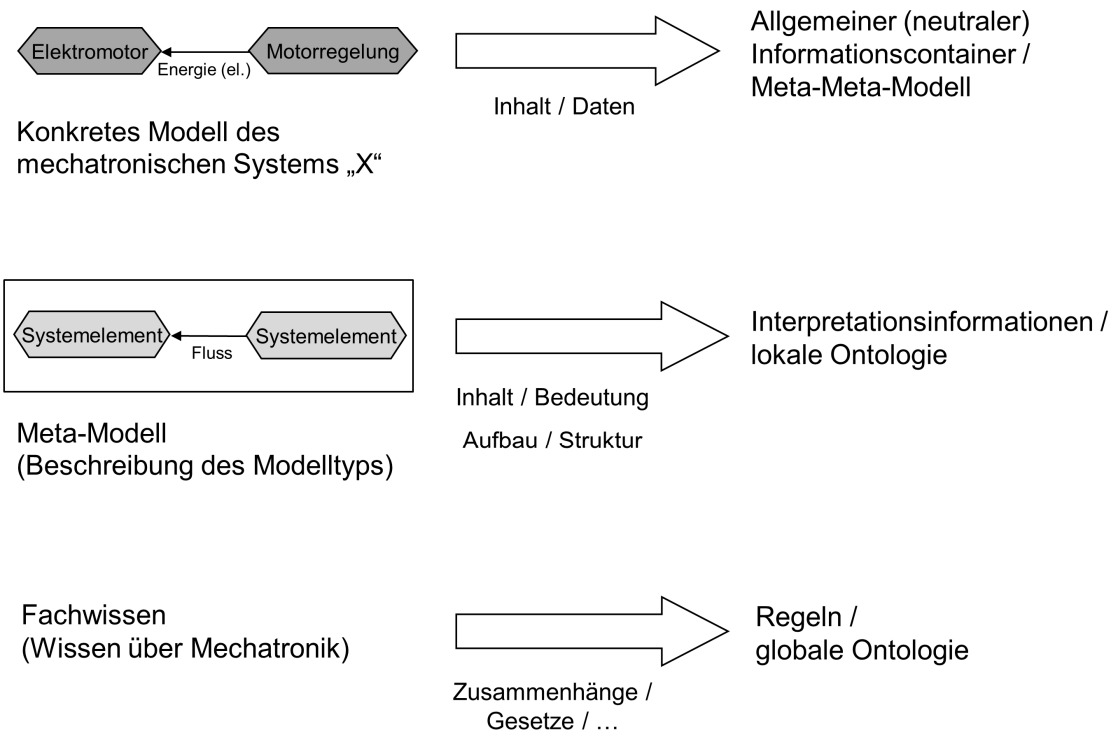


Abbildung 6.5: Voraussetzungen für eine rechnergestützte Prüfung heterogener Modelle

In Abbildung 6.5 sind die für eine rechnergestützte Prüfung heterogener Modelle notwendigen Voraussetzungen zusammengefasst. Die Informationen aus einem konkreten, aber beliebigen Modell müssen einheitlich abgebildet werden können. Informationen zur Interpretation des Modellinhalts müssen vorhanden sein. Das Wissen über mechatronische Zusammenhänge und Fakten muss zur Verfügung stehen, damit entsprechende Regeln abgeleitet und auf die Modelle angewendet werden können.

## 6.4 Abstraktion der Entwurfsmodelle

Die Modelle bzw. deren Beschreibungssprachen müssen soweit formalisiert und abstrahiert werden, dass das Auslesen der Modelle und deren Verarbeitung mit Hilfe eines Softwaresystems möglich wird. Beispielsweise müssen bei graphbasierten Modellen alle einsetzbaren Elemente sowie deren erlaubte Verknüpfungen definiert werden. Dabei ist die Darstellung oder Symbolik (z. B. als Raute, Rechteck etc.) zweitrangig. Wichtig ist, dass alle Modelle für Mensch und Softwaresystem eindeutig interpretierbar sind. Ein Beispiel für eine klar definierte Beschreibungssprache ist im Bereich der Softwaretechnik die UML [OMG11], die sich dort zum Standard entwickelt hat. Mit ihren verschiedenen Beschreibungsmitteln wie Klassen-, Sequenz- oder Zustandsdiagrammen ist die UML in der Lage, unterschiedliche Modelle, die bei einer Softwareentwicklung benötigt werden, darzustellen. Dabei definiert das Klassendiagramm, welche Typen von Objekten in der späteren Software existieren werden und welche Eigenschaften und Beziehungen diese besitzen dürfen. Die in einem Klassendiagramm ebenfalls abgebildeten Methoden werden im Folgenden jedoch nicht berücksichtigt, da lediglich passive Informationscon-

tainer gebildet werden sollen. Im weiteren Verlauf werden die mechatronischen Modelle soweit abstrahiert, bis sie in Form von Klassendiagrammen darstellbar sind und somit in einem Softwaresystem abgebildet werden können. Im folgenden Unterkapitel wird dies beispielhaft am Modell der Wirkstruktur der Partialmodelle nach [GFDK09] durchgeführt. Am Ende dieses Abstraktionsschrittes steht also eine formale Beschreibung des Inhalts jedes Modells zur Verfügung. Eine Übersicht über diese Abstraktionsebenen ist in Abbildung 6.6 dargestellt.

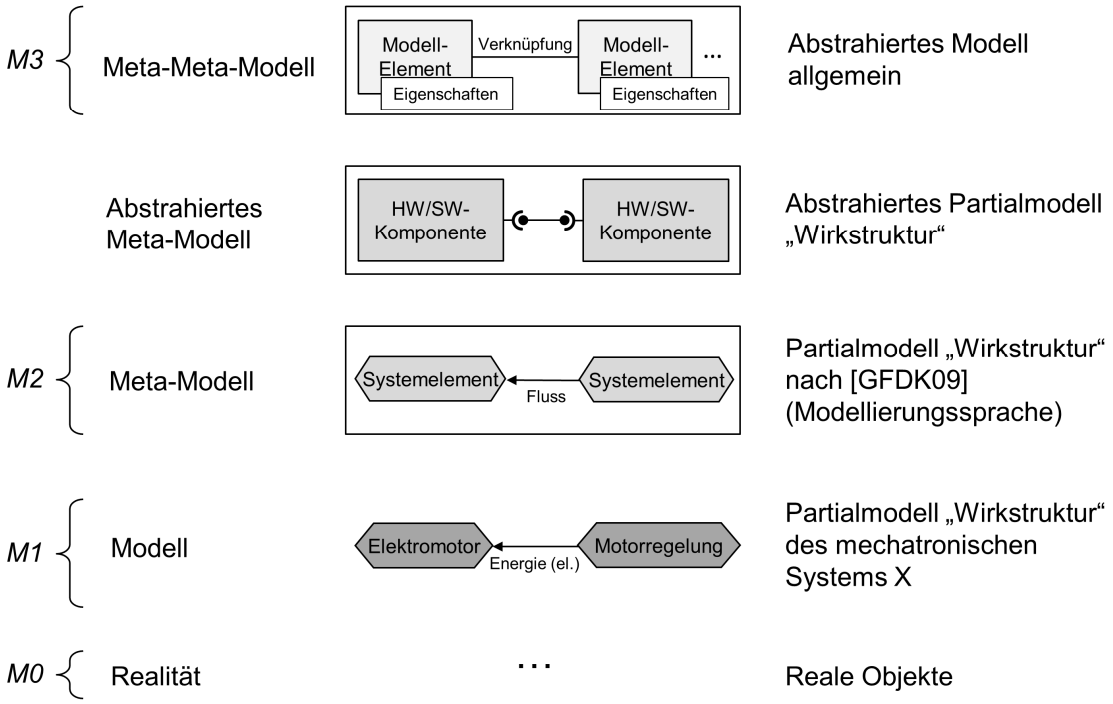


Abbildung 6.6: Abstraktion der mechatronischen Modelle

Dabei ist das Modell der untersten Ebene zugeordnet und bezeichnet die Modellierung eines konkreten mechatronischen Systems. In der Literatur wird diese Ebene als „M1“ bezeichnet [OMG11], wobei die darunterliegende Ebene „M0“ die realen Objekte (Instanzen) enthält. Über dem konkreten Modell befindet sich die Modellierungssprache, das Meta-Modell (Lit: „M2“). Sie bestimmt den Aufbau bzw. den Modelltyp des darunterliegenden Modells und definiert die benutzbaren Symbole bzw. die Notation. Das abstrahierte Meta-Modell stellt die Übersetzung des Meta-Modells in ein UML-Klassendiagramm dar. Zusätzlich wird eine Abstraktion des Modells durchgeführt, um weitere Notationsformen (z. B. SysML [OMG12]) zu unterstützen. Diese wird hier als abstrahiertes Meta-Modell bezeichnet, obwohl in gewisser Weise bereits von einem Meta-Meta-Modell gesprochen werden könnte. Um mit [OMG11] konform zu bleiben, wird dieser Begriff erst in der nächsten Stufe benutzt und diese Abstraktion lediglich als Zwischenschritt angesehen.

In einem weiteren Schritt ist es notwendig, die abstrahierten Meta-Modelle auf die nächste Meta-Ebene zu heben, um alle möglichen Modelltypen formal darstellen zu können. Die das Meta-Meta-Modell enthaltene Ebene wird auch als „M3“ bezeichnet und wird für die UML mit Hilfe

der MOF (Meta Object Facility) [OMG14] definiert. Deshalb wird das Meta-Meta-Modell im Rahmen dieser Arbeit ebenfalls gemäß der MOF erstellt. Erst die Einführung einer formalen und allgemeinen Beschreibungssprache für alle Modelle macht es möglich, diese automatisiert zu verarbeiten und auf Konsistenz zu prüfen. Dieses gemeinsame Meta-Meta-Modell dient als abstrakter Informationscontainer, schafft die Verbindung zwischen beliebigen Modellen und macht diese, mit Unterstützung weiterer Informationen, interpretierbar. Im Folgenden wird deshalb dieses Meta-Meta-Modell mit Hilfe der Formalisierung eines beispielhaften Modells entwickelt.

### 6.4.1 Formalisierung am Beispiel „Wirkstruktur“

Das Modell Wirkstruktur der Partialmodelle nach [GFDK09] (siehe Kapitel 2.3.3) beschreibt die im späteren mechatronischen System vorhandenen Komponenten und ist damit für Personen, die nicht an der Entwicklung beteiligt sind, vergleichsweise gut verständlich. Es wird hier deshalb als Beispiel zur Formalisierung herangezogen. Die Partialmodelle werden in [Fran06] ausführlich vorgestellt, bedingt durch die Weiterentwicklung in weiteren Veröffentlichungen, z. B. [GSG+09], in leicht veränderter Form präsentiert. Hier liegt die Konzentration deshalb auf den Bereichen, die sich nicht oder wenig verändert haben. Mögliche Weiterentwicklungen werden aufgezeigt und bei der Formalisierung berücksichtigt.

Zum besseren Verständnis wird ein Beispiel eines realen mechatronischen Systems benutzt, welches in den Modellen jeweils dargestellt wird. In Abbildung 6.7 ist der Ausschnitt einer Antriebseinheit des „Automatisierten Fußballschuhs David“ [RJG11] abgebildet. Sie besteht aus einem Elektromotor, der über ein Getriebe ein Transportband (nicht dargestellt) antreibt. Der Motor wird über eine Motorregelung mit Energie versorgt, die wiederum von einer Geschwindigkeitsregelung gesteuert wird. Mit Hilfe einer Drehzahlmessung wird der Regelkreis geschlossen, indem die Geschwindigkeitsregelung mit der Ist-Drehzahl versorgt wird.

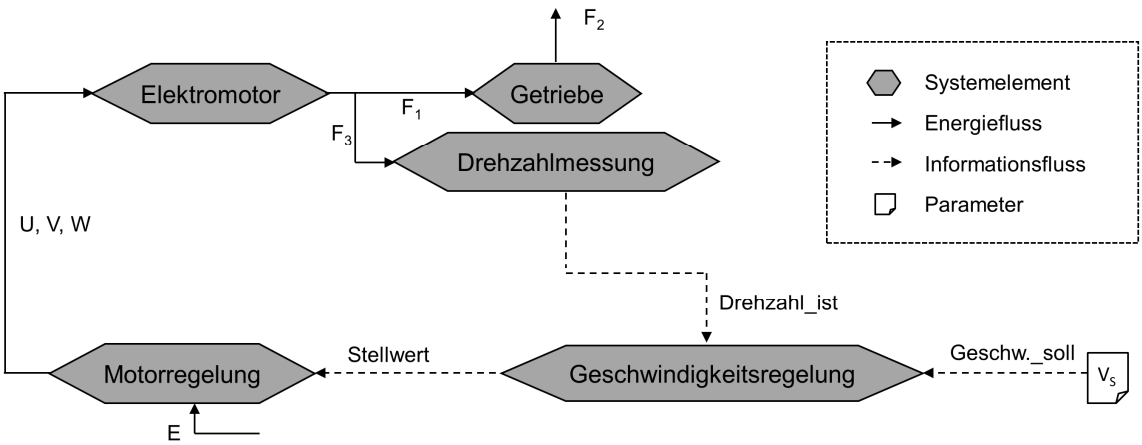


Abbildung 6.7: Wirkstruktur einer Antriebseinheit nach [GFDK09]

Die Systemelemente werden mit Hilfe eines Sechsecks dargestellt und erhalten eine Bezeichnung. Flüsse werden durch Pfeile repräsentiert. Ein durchgezogener Pfeil bezeichnet einen

Energiefluss, ein gestrichelter Pfeil einen Informationsfluss. Ein eingehender Parameter wird durch ein geknicktes Rechteck symbolisiert. Zudem können beispielsweise Störeinflüsse, Parameter, Ziele usw. (entsprechen Verknüpfungen zu anderen Modellen) dargestellt werden. Weitere Symbole können [Fran06] oder [GFDK09] entnommen werden.

In Abbildung 6.8 werden die Systemelemente um Ports (Ein- und Ausgänge) erweitert und Energie-, Material- und Informationsflüsse zum allgemeinen Fluss abstrahiert. Die Erweiterung um Ports unterstützt die Verknüpfung der Systemelemente durch Flüsse und ein Systemelement kann einen oder mehrere Ports besitzen.

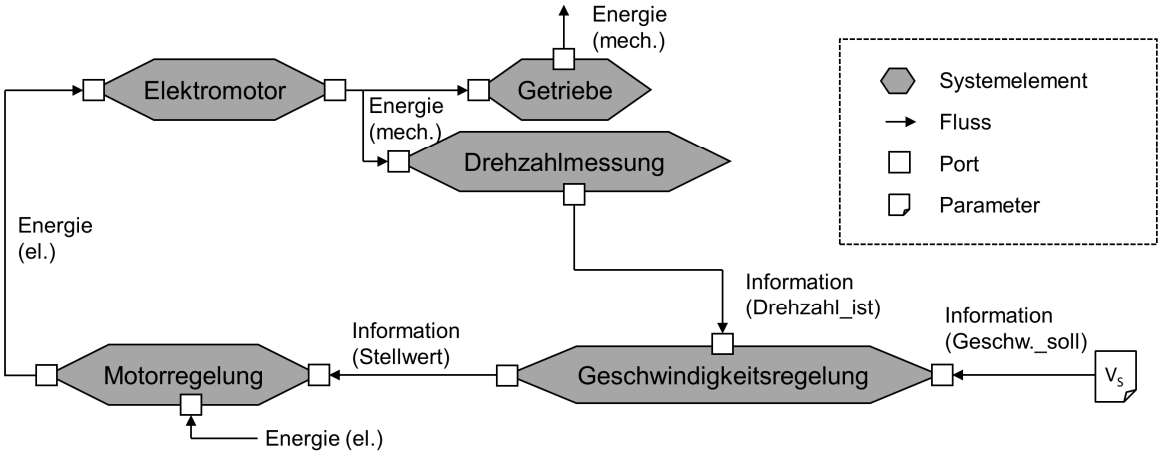


Abbildung 6.8: Erweiterte Wirkstruktur einer Antriebseinheit

Alle Modellelemente besitzen Eigenschaften, mit denen sie sich näher spezifizieren lassen (in Abbildung 6.8 nur bei den Flüssen dargestellt). Damit ergibt sich die Möglichkeit, die Elemente im Lauf der Entwicklung zu detaillieren. So kann ein einfacher Motor beispielsweise in einzelnen Schritten zu einem Elektromotor und später zu einer Asynchronmaschine verfeinert werden. Ebenso können Energieflüsse über elektrische Energieflüsse zu Energieflüssen mit genauen Spannungs- und Stromwerten spezifiziert werden.

Für die folgenden Abstraktionen wird nur ein Ausschnitt der Wirkstruktur, bestehend aus zwei Systemelementen, verwendet. Im ersten Schritt werden diese Systemelemente zu HW/SW-Komponenten abstrahiert [RaGö12b, RaGö13a]. In Abbildung 6.9 werden zum besseren Verständnis für die jeweils allgemeinsten Elemente Symbole aus der UML (Rechteck und Schnittstelle) verwendet, abgeleitete Elemente sind lediglich textuell dargestellt. Eine HW/SW-Komponente kann dabei eine elementare Komponente oder eine zusammengesetzte Komponente darstellen. So ist es beispielsweise möglich, Blockdiagramme in SysML [OMG12] ebenfalls abzubilden. Die Ports werden zu Schnittstellen, die Anknüpfpunkte für Verbindungen sind, abstrahiert. Eine Verbindung ist die abstrahierte Form eines Flusses und kann somit einen Fluss oder eine modellierte Abhängigkeit repräsentieren. Mit Hilfe von Abhängigkeiten können über Flüsse hinausgehende Informationen im Modell abgelegt werden, wie beispielsweise die Information, dass eine Komponente eine andere benötigt oder das zwei gleichartige Komponenten

unterschiedlich realisiert werden müssen (diversitäre Redundanz). Auch in dieser abstrahierten Meta-Ebene besitzen alle Elemente Eigenschaften, mit denen sich diese genauer spezifizieren lassen (in Abbildung 6.9 nicht dargestellt)

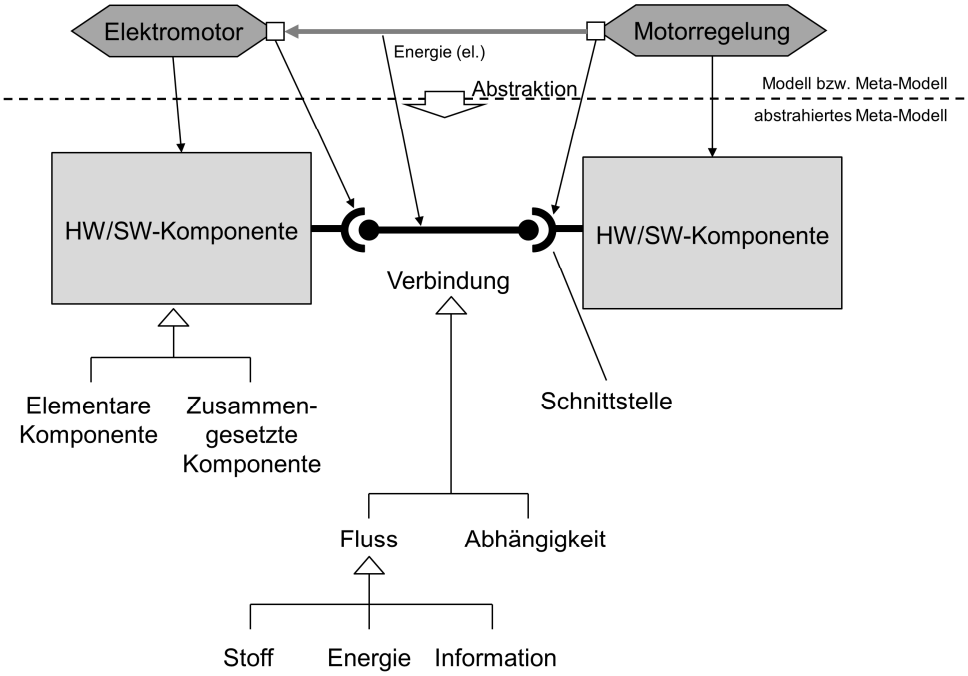


Abbildung 6.9: Entstehung des abstrahierten Meta-Modells der Wirkstruktur

Im zweiten Schritt wird das abstrahierte Meta-Modell abermals abstrahiert und das Meta-Meta-Modell erstellt. Mit diesem Meta-Meta-Modell lassen sich alle Modelle, nicht nur die Wirkstruktur beschreiben. In Abbildung 6.10 ist zu sehen, dass alle Elemente des abstrahierten Meta-Modells (HW/SW-Komponente, Schnittstelle und Verbindung) zu Modellelementen verallgemeinert werden.

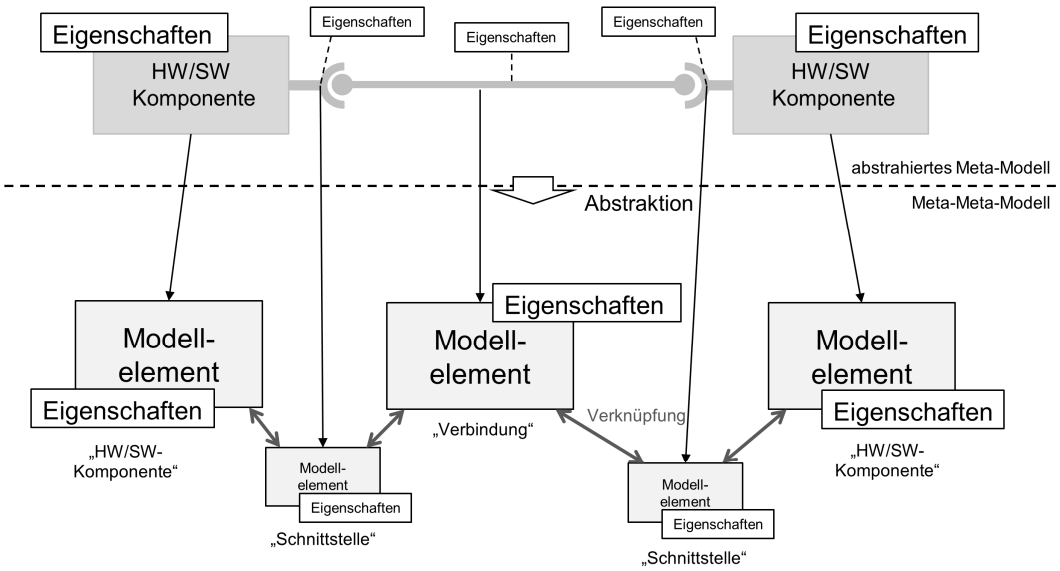


Abbildung 6.10: Entstehung des Meta-Meta-Modells

Diese Modellelemente besitzen ebenfalls Eigenschaften zu deren genaueren Spezifikation. So kann beispielsweise die Klasse (der Typ) eines Modellelements in einer Eigenschaft festgehalten werden (z. B. „HW/SW-Komponente“ oder „Fluss“). Eigenschaften bestehen dabei aus einer Bezeichnung, einem Typ (z. B. Integer) und einem Wert. Zwischen den einzelnen Modellelementen können Verknüpfungen bestehen. Diese besitzen selbst keine Eigenschaften sondern zeigen lediglich die Zusammengehörigkeit von Modellelementen an.

Modellelemente können beliebig verknüpft werden, es sind also Netze von Modellelementen möglich. Abbildung 6.11 zeigt ein solches Netz aus Modellelementen auf Meta-Meta-Ebene.

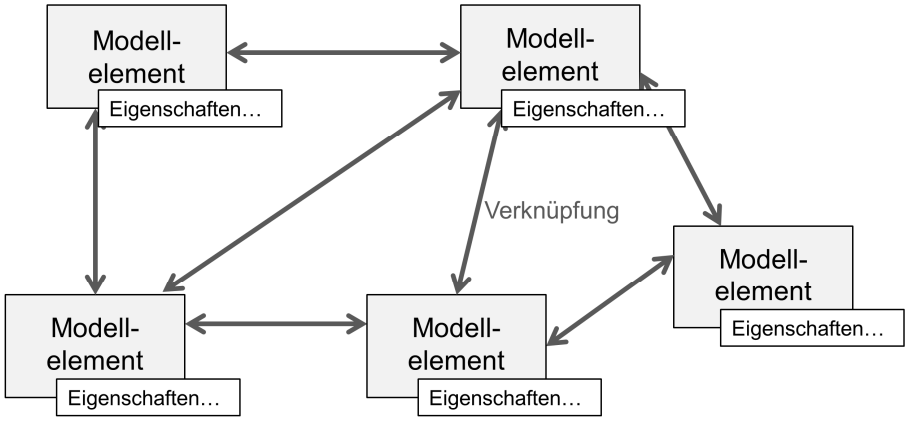


Abbildung 6.11: Allgemeines Modell auf Meta-Meta-Ebene

Um ein Meta-Meta-Modell in Software abzubilden, muss es in ein Klassendiagramm (der UML) übersetzt werden. Damit ist der Aufbau definiert und die einzelnen Klassen können in Form von Informationscontainern realisiert werden. Abbildung 6.12 zeigt das entsprechende Klassendiagramm.

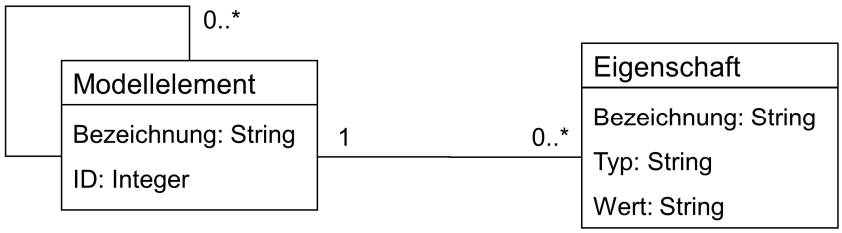


Abbildung 6.12: Klassendiagramm des Meta-Meta-Modells in UML

Die Klasse Modellelement kann beliebig viele Verknüpfungen mit anderen Modellelementen eingehen. Sie besitzt jedoch lediglich zwei feste Attribute Bezeichnung und ID, um es zu identifizieren. Weitere Details eines Modellelements werden in dessen Eigenschaften festgelegt, von denen jedes Modellelement ebenfalls eine beliebige Anzahl besitzen kann. Eine Eigenschaft gehört lediglich zu einem Modellelement.

## 6.4.2 Erprobung an weiteren Modellen

In diesem Abschnitt wird exemplarisch gezeigt, dass das Meta-Meta-Modell auch auf weitere Modelle anwendbar ist. Modelle, die der Wirkstruktur ähnlich sind, sind ohne weiteres durch das Meta-Meta-Modell repräsentierbar. Ein UML-Klassendiagramm ist ein Beispiel hierfür. Hier gibt es ebenfalls Elemente (Klassen, Attribute und Methoden), die auf unterschiedliche Art und Weise miteinander verknüpft sind. Für andere Modelle, wie den Anforderungen an ein System, ist die Abbildbarkeit nicht so einfach ersichtlich. Im Folgenden wird die Abbildbarkeit deshalb an verschiedenen Beispielen gezeigt.

### Modell Anforderungen

Die Anforderungen werden in der Regel in Tabellenform dokumentiert. Abhängig von der Definition der Anforderungsliste besitzt die entstehende Tabelle unterschiedliche Spalten. Den meisten Beschreibungen gemein sind die Spalten Nummerierung, Bezeichnung, Beschreibung der Anforderung und Wichtigkeit (Forderung/Wunsch). Hinzukommen können ein Änderungsdatum, eine Versionierung, die ändernde Person oder die Quelle der Anforderung. Ausgehend von der einfachsten Form, kann die Anforderungsliste wie in Tabelle 6.1 dargestellt aussehen.

Tabelle 6.1: Beispielhafte Anforderungsliste

| **Nr.** | **Bezeichnung** | **Beschreibung** | **F/W** |
|---|---|---|---|
| 1.1 | Gewicht | Das Gewicht der Maschine muss unter 750 kg liegen. | Forderung |
| 1.2 | Kosten | Die Entwicklungskosten müssen unter 20.000 € sein. | Wunsch |
| 1.3 | Durchsatz | Der Durchsatz muss bei mind. 45.000 Stück/h liegen. | Forderung |
| … | … | … | … |

Die Repräsentation dieser Tabelle gemäß dem Meta-Meta-Modell kann beispielsweise wie folgt realisiert werden. Ein Modellelement repräsentiert die Anforderungsliste selbst, jede Anforderung (Zeile der Tabelle) wird ebenfalls durch jeweils ein Modellelement repräsentiert. Die Eintragungen der einzelnen Tabellenspalten werden durch Eigenschaften repräsentiert, wobei die Bezeichnung der Eigenschaft die Bezeichnung der Spalte wiedergibt. So können Tabellen mit beliebiger Zeilen- und Spaltenanzahl mit Hilfe des Meta-Meta-Modells abgebildet werden.

### Modell Verhalten

Das Verhalten von mechatronischen Systemen oder von Systemen allgemein kann beispielsweise mit Hilfe von Zustands- und Sequenzdiagrammen beschrieben werden. Diese sind in der UML [OMG11] definiert. Abbildung 6.13 zeigt links ein Zustandsdiagramm und rechts dessen Repräsentation im Meta-Meta-Modell.

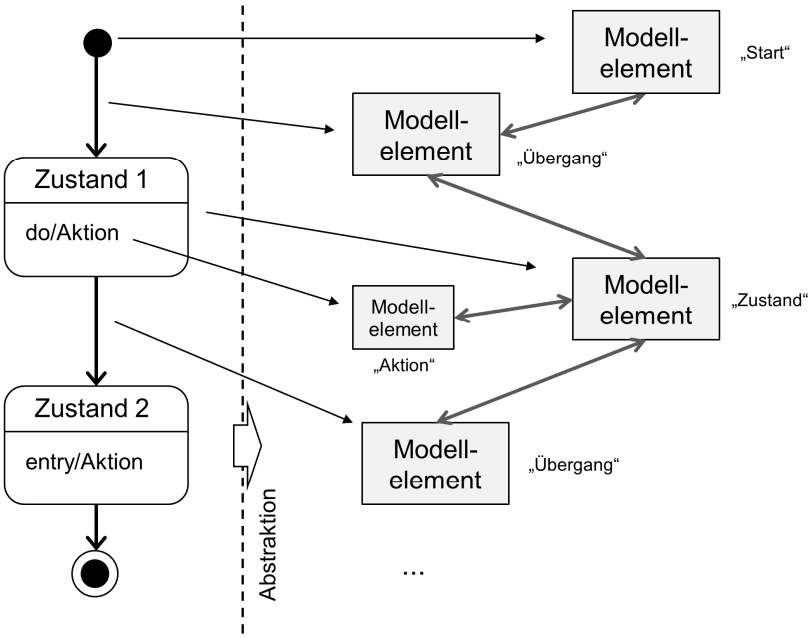


Abbildung 6.13: Beispiel für ein Zustandsdiagramm und dessen Abbildung im Meta-Meta-Modell

Für das Zustandsdiagramm werden Start, Ende und Zustände sowie Entry-, Do- und Exit-Aktionen durch Modellelemente repräsentiert. Eigenschaften helfen Detailinformationen zu ergänzen und die Aktionen zu unterscheiden. Zustandsübergänge und Übergangsbedingungen werden ebenfalls durch Modellelemente abgebildet. Textuelle Beschreibungen können in Form von Eigenschaften modelliert werden. Auf ähnliche Art und Weise können auch Aktivitätsdiagramme oder Petri-Netze, die ebenfalls Zustände bzw. Aktivitäten und Übergänge besitzen, modelliert werden.

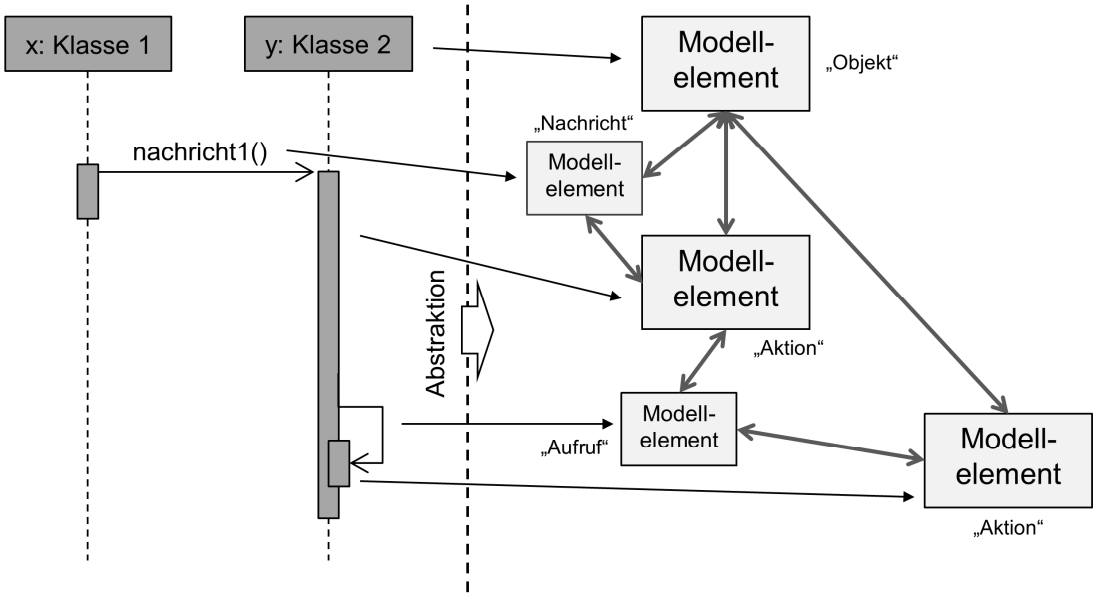


Abbildung 6.14: Beispiel für ein Sequenzdiagramm und dessen Abbildung im Meta-Meta-Modell

Ein Sequenzdiagramm (siehe Abbildung 6.14) besteht aus Objekten, deren Aktivitäten und Nachrichten bzw. Aufrufen. Diese werden mit Hilfe von Modellelementen abgebildet. Eigenschaften unterstützen die Modellierung der (zeitlichen) Position der einzelnen Elemente.

### Modell Gestalt

Heutzutage werden in der Regel CAD-Systeme (Computer-Aided-Design-Systeme) eingesetzt, um zwei- oder dreidimensionale Modelle von einzelnen Bauteilen oder Baugruppen zu erstellen. So kann mit relativ wenig Aufwand eine Vorstellung des späteren Systems gewonnen werden. Auch diese Modelle lassen sich mit Hilfe des Meta-Meta-Modells modellieren. Dazu werden alle benötigten Elemente des Modells als Modellelemente repräsentiert. Abbildung 6.15 zeigt ein einfaches Beispiel, in dem zwei dreidimensionale Bauteile (Welle und Nabe) miteinander verbunden wurden.

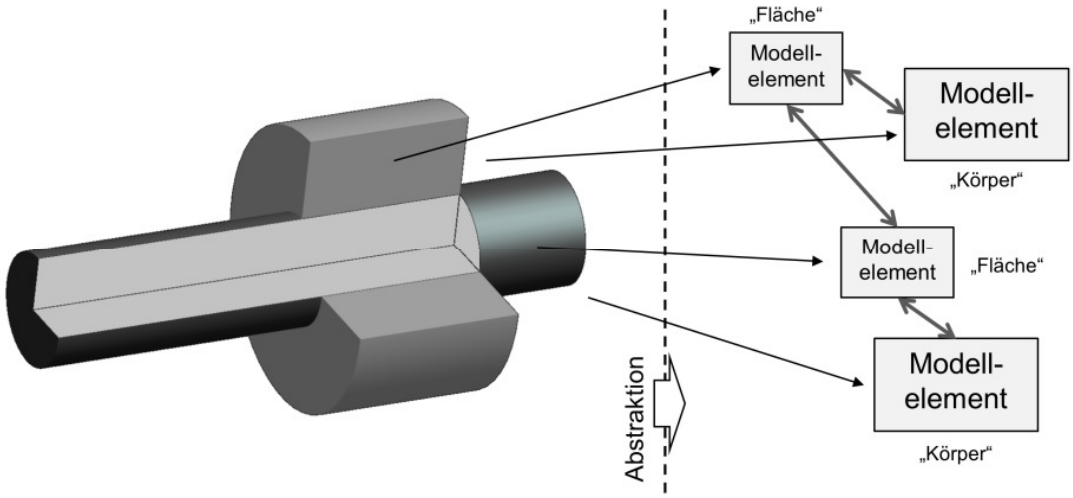


Abbildung 6.15: Dreidimensionales CAD-Modell und dessen Abbildung im Meta-Meta-Modell

Für die Repräsentation der Bauteile werden mehrere Modellelemente benötigt. Das Bauteil selbst bzw. dessen Körper wird mit Hilfe eines Modellelements abgebildet. In den Eigenschaften kann die geometrische Form des Körpers abgelegt werden. Kommen komplexere Formen zum Einsatz, so sind diese aus mehreren elementaren Geometrien und damit aus mehreren Modellelementen zusammenzusetzen. Die Oberflächen eines Körpers werden ebenfalls mittels Modellelementen repräsentiert (in Abbildung 6.15 nur für beteiligte Oberflächen dargestellt). So ist es möglich, Körper mit Hilfe ihrer Oberflächen zu verbinden. Genauso können beispielsweise auch die Kanten eines Körpers durch Modellelemente abgebildet werden, sofern diese für die Anwendung relevant sind.

### 6.4.3 Grenzen der Verallgemeinerung

Das Meta-Meta-Modell ist universell einsetzbar und beliebige Modelle können damit beschrieben werden. Trotzdem sind der Formalisierung Grenzen gesetzt. Es lassen sich zwar prinzipiell alle Modelle beschreiben, jedoch schwanken der Aufwand und der Umfang des abstrahierten Modells beträchtlich. Mit der Größe der Beschreibung auf Meta-Meta-Ebene steigen der Aufwand und die Komplexität der Interpretation des Modells.

Der Vergleich zwischen einer Tabelle und einem dreidimensionalen CAD-Modell zeigen diesen Unterschied auf. Die Tabelle, die in der Meta-Meta-Ebene eine Baumstruktur aufweist (Wurzel ist die Tabelle selbst, Zweige sind die einzelnen Zeilen), ist relativ einfach zu interpretieren und zu durchsuchen. Im CAD-Modell wird ein einzelner Zylinder bereits durch 4 Modellelemente repräsentiert (Körper + 3 Flächen). Werden nun zwei Zylinder unterschiedlicher Durchmesser koaxial miteinander verbunden, benötigt man bereits 7 Modellelemente (2 Körper + 5 Flächen + 2 versteckte Flächen). Da in einem CAD-System in der Regel frei konstruiert werden kann, sind beliebige Kombinationen möglich. Damit sind zur Repräsentation sehr viele Modellelemente notwendig und die Interpretation ist auf Grund der unzähligen Kombinationsmöglichkeiten sehr aufwendig. Mit dem Einsatz von Ontologien wurden Fortschritte auf diesem Gebiet möglich, es ist jedoch auch heute noch Gegenstand der Forschung [GGB11].

In diesem Kapitel wurden das prinzipielle Vorgehen beim Umgang mit heterogenen Modellen und das dieser Arbeit zugrunde liegende Konzept vorgestellt. Modelle werden dazu mit Hilfe einer Modellbeschreibung auf eine modellunabhängige Ebene abstrahiert und dort mit Hilfe von Regeln geprüft. Das dazu benötigte Fachwissen sowie die Modellbeschreibungen liegen in Form von Ontologien vor. Die Prüfung selbst wird mit Hilfe von Softwareagenten durchgeführt. Die Voraussetzungen zur Umsetzung des Konzepts sind neben den Regeln zur Prüfung die Möglichkeit der Abstraktion der Modelle, wozu Interpretationsinformationen (Modellbeschreibung) und modellneutrale Informationscontainer notwendig sind. Am Beispiel der Wirkstruktur wurde die Formalisierung präsentiert, indem das Modell über das abstrahierte Meta-Modell zum Meta-Meta-Modell abstrahiert wurde. Unterschiedliche Beispiele halfen, die Abbildbarkeit verschiedenster Modelle im Meta-Meta-Modell zu zeigen. Abschließend muss festgehalten werden, dass zwar alle Modelle in das Meta-Meta-Modell verallgemeinert werden können, die Interpretation dieser abstrahierten Modelle jedoch derzeit noch nicht für alle Modelle (z. B. frei konstruierte CAD-Modelle) realisiert werden kann. Wie die Modelle geprüft werden, wird im folgenden Kapitel vorgestellt.

# 7 Konzept der agentenbasierten Prüfung

Die Prüfung der Modelle soll von einem Agentensystem durchgeführt werden. Agenten eignen sich, wie in Kapitel 3 beschrieben, sehr gut, um komplexe und verteilte Problemstellungen flexibel zu lösen. Um ein Agentensystem zu entwickeln, wird zunächst ein Überblick über die vorhandenen Entitäten gegeben. Im nächsten Schritt werden die von den Agenten auszuführenden Aufgaben (Rollen) definiert. Ist dies geschehen, so werden die Rollen im zweiten Schritt zu Agententypen zusammengefasst und diese vorgestellt. Anschließend wird der Aufbau der Agenten im dritten Schritt erläutert. Schließlich werden die Interaktionen zwischen den Agenten im vierten Schritt näher beleuchtet.

## 7.1 Übersicht über die im System vorhandenen Entitäten

Bevor mit der Entwicklung des Agentensystems begonnen werden kann, muss ein Überblick über die für die Prüfung der Modelle benötigten Entitäten geschaffen werden. Neben der globalen Ontologie sind in Abbildung 7.1 der Benutzer und die zu prüfenden Regeln dargestellt. In der darunterliegenden Ebene befinden sich n heterogene Modelle. Jedes Modell besitzt eine lokale Ontologie, die die Modellbeschreibung enthält.

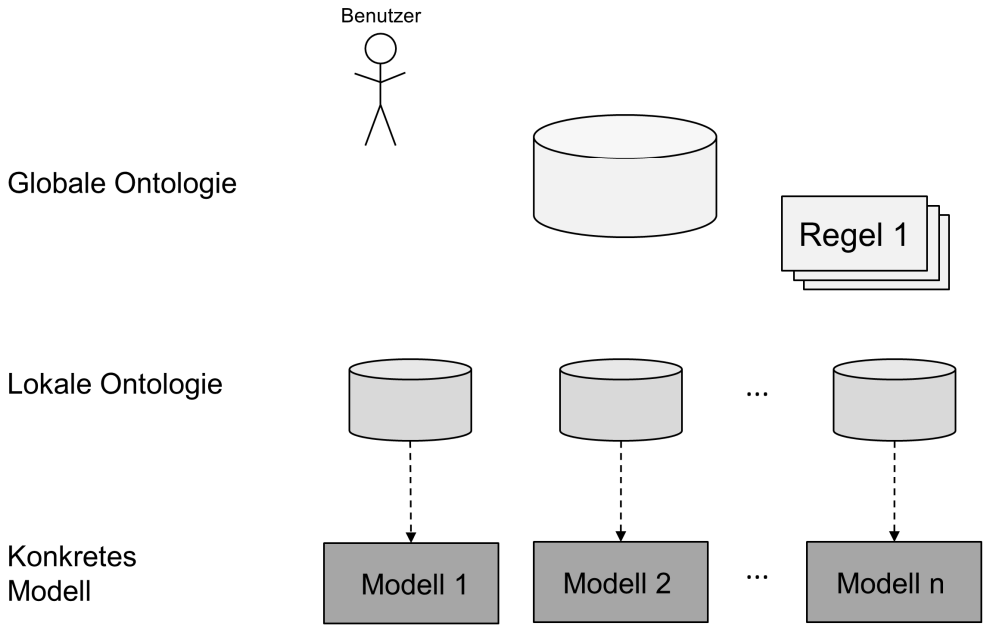


Abbildung 7.1: Überblick über die für die Prüfung der Modelle benötigten Entitäten

Ziel der Prüfung ist es, Inkonsistenzen sowohl innerhalb eines einzelnen Modells als auch zwischen mehreren Modellen aufzudecken. Dazu werden entsprechende Regeln aus der globalen Ontologie abgeleitet und sind damit unabhängig von den einzelnen Modellen. Die lokalen Ontologien sind notwendig, um die heterogenen Modelle in die globale und somit einheitliche Be-

griffswelt zu übersetzen bzw. die in der globalen Begriffswelt formulierten Regeln auf die lokalen Modelle anzuwenden. Dem Benutzer werden schließlich die Ergebnisse der Prüfung präsentiert.

Die Modelle wurden bisher abstrakt als Modelle betrachtet. Für die Entwicklung des Agentensystems sind jedoch weitere Details erforderlich. Ein Modell wird in einem Engineering Werkzeug (Editor) erstellt und gepflegt. Diese besitzen häufig eine Datenschnittstelle, über die auf das Modell im Werkzeug zugegriffen werden kann. Existiert keine solche Schnittstelle, so kann man beispielsweise direkt über die gespeicherte Datei des Modells an die jeweiligen Informationen gelangen. Abbildung 7.2 zeigt das in ein Engineering Werkzeug eingebettete Modell und die zugehörige Datenschnittstelle.

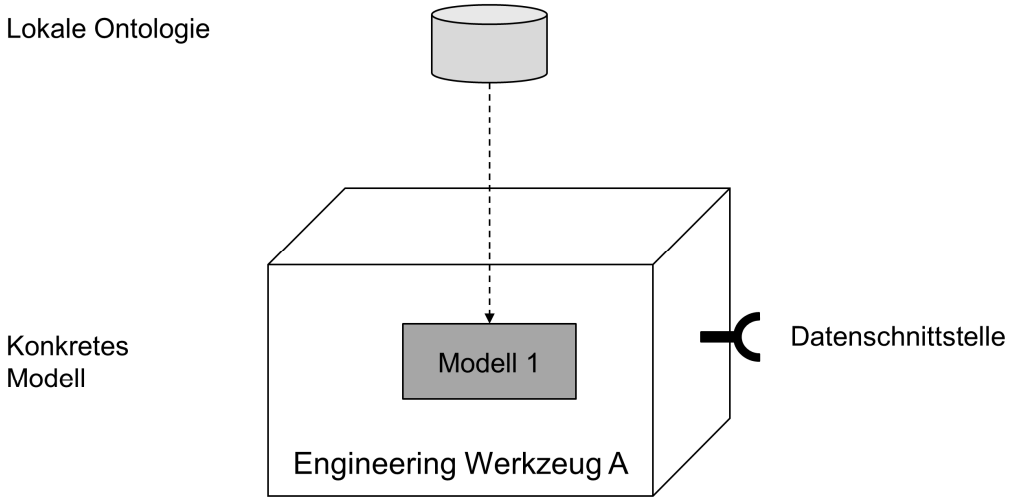


Abbildung 7.2: Modell im Engineering Werkzeug

Die Datenschnittstellen der Werkzeuge erlauben teilweise die automatisierte Überwachung der Modelle auf Änderungen durch den Benutzer und ermöglichen so eine permanente Prüfung der Modelle, für den Benutzer unsichtbar und im Hintergrund. Erst wenn eine Inkonsistenz festgestellt wird, wird der Benutzer informiert, der dieses Problem sofort beheben kann. Eine solche Überwachung ist beispielsweise in [KRBG11a] vorgestellt. Bei der Betrachtung von heterogenen Modellen muss jedoch auch von heterogenen Engineering Werkzeugen und somit von unterschiedlichen Datenschnittstellen ausgegangen werden. Hier zeigen sich die Vorteile des Einsatzes von Agenten sehr deutlich. Durch die lose Kopplung der Agenten über Kommunikationsprotokolle, können die Modellagenten für verschiedene Modelle intern sehr unterschiedlich realisiert und trotzdem zur Laufzeit des Systems integriert werden.

## 7.2 Aufgaben bzw. Rollen der Agenten

Mit diesem Wissen kann zur Ermittlung der Aufgaben bzw. Rollen der Agenten übergegangen werden. Die Aufgaben beschreiben dabei, welche Tätigkeiten im Agentensystem durchgeführt

werden müssen. Jede Aufgabe wird dabei von einer Rolle übernommen, die anschließend den verschiedenen Agententypen zugeordnet werden. Ein einzelner Agententyp kann dabei mehrere Rollen wahrnehmen. Abbildung 7.3 zeigt eine Übersicht der Rollen, wobei diese bereits zu Gruppen verdichtet wurden.

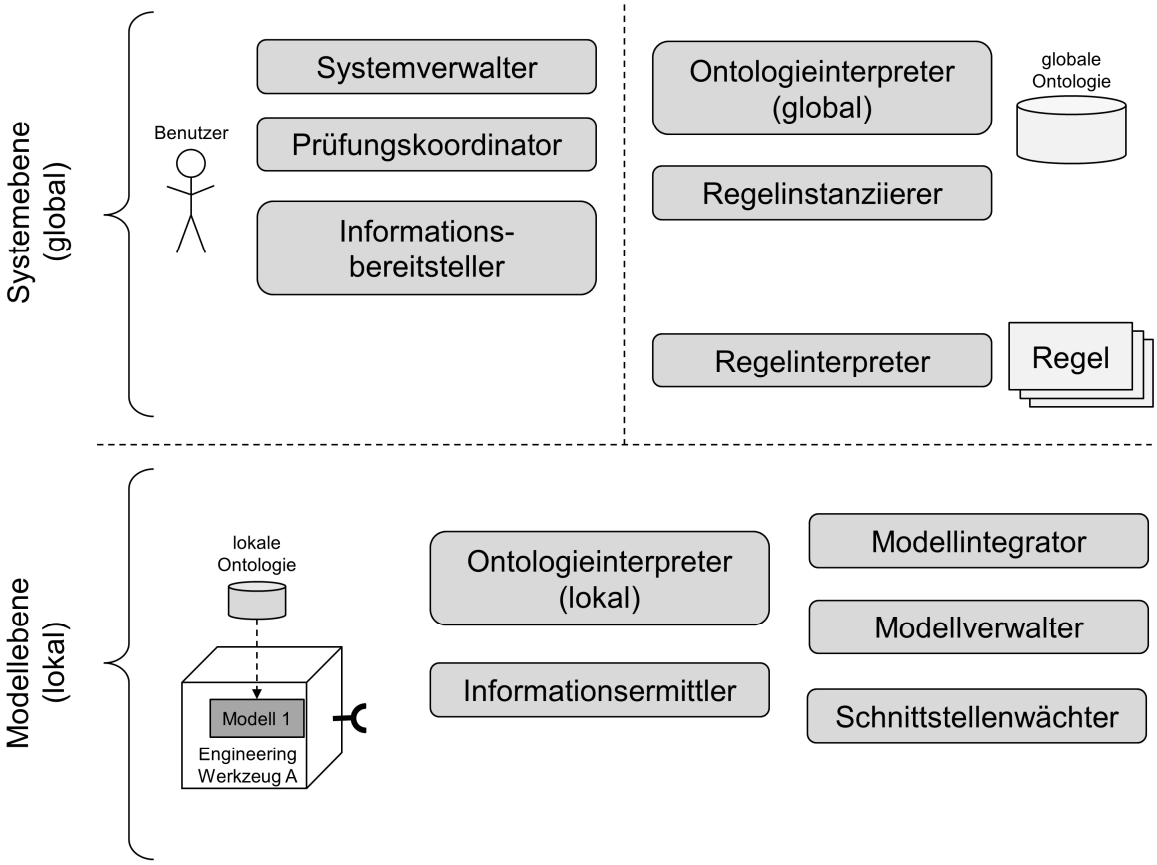


Abbildung 7.3: Gruppierte Rollen der Agenten

Die obere Hälfte der Abbildung zeigt die globalen Rollen, die systemweit ausgeführt werden müssen. Auf der linken Seite befinden sich die Rollen, die die Verwaltung der Modelle betreffen oder Interaktionen mit dem Benutzer erfordern. Auf der rechten Seite sind die Rollen, die eng mit dem globalen Wissen verzahnt sind. Sie betreffen die globale Ontologie sowie die Regeln, die zusammen das Wissen im System darstellen.

In der unteren Hälfte der Abbildung sind die Rollen auf der lokalen Modellebene dargestellt. Sie existieren für jedes Modell und sind deshalb diesen zugeordnet.

In der folgenden Auflistung sind die einzelnen Rollen und ihre Aufgaben detaillierter beschrieben:

- *Systemverwalter:* Der Systemverwalter verwaltet das System der Modelle. Bei ihm müssen neue Modelle angemeldet werden und er weiß, welche Modelle im System vorhanden sind und wer für ein Modell zuständig ist und Informationen daraus bereitstellen kann.

- *Prüfungskoordinator:* Zur Koordination einer Prüfung gehört neben der Beauftragung und Überwachung der Prüfer auch die Entscheidung über das Ende sowie das Ergebnis der Prüfung. Bei Eingang eines Prüfungsauftrags wird zunächst eine Liste der im System vorhandenen Modelle vom Systemverwalter angefordert. Anschließend werden die zuständigen Modellprüfer mit der Prüfung beauftragt und die Ergebnisse empfangen. Die Ergebnisse enthalten neben den verletzten Regeln die zur Prüfung der jeweiligen Regel verwendeten Informationen und deren Ursprung im Modell. Sind die Ergebnisse aller Prüfer eingetroffen, so kann die Prüfung als beendet erklärt werden.

- *Informationsbereitsteller:* Die Informationen, die bei einer Prüfung gesammelt werden, werden vom Informationsbereitsteller aufbereitet und dem Benutzer als Unterstützung bei der Behebung der aufgedeckten Inkonsistenzen bereitgestellt. Er stellt zudem die Schnittstelle zum Benutzer dar.

- *Ontologieinterpreter (global):* Die globale Ontologie ist die zentrale Wissensbasis der Modellprüfung. Der Ontologieinterpreter führt Anfragen an die Ontologie aus und stellt deren Wissen bereit. Er dient somit als Schnittstelle zur globalen Ontologie.

- *Regelinstanziierer:* Der Regelinstanziierer leitet diejenigen Regeln aus dem Wissen der globalen Ontologie ab, die dort nicht explizit formuliert sind. Dazu greift er auf den Ontologieinterpreter zurück. Für explizite und abgeleitete Regeln instanziiert er jeweils zugehörige Regelinterpreter, die die jeweilige Regel später anwenden.

- *Regelinterpreter:* Der Regelinterpreter prüft, ob die ihm zugeordnete Regel von einem Modell oder einer Gruppe von Modellen erfüllt ist. Informationen aus den Modellen, die bei dieser Prüfung benötigt werden, fragt der Regelinterpreter beim jeweils zuständigen Informationsermittler an. Die Ergebnisse der Prüfung werden an den Prüfungskoordinator gemeldet. Nichterfüllte Regeln werden, zusammen mit den Informationen aus den Modellen, die zur Nichterfüllung geführt haben, ebenfalls gemeldet.

- *Ontologieinterpreter (lokal):* Die lokalen Ontologien sind die Bindeglieder zwischen den heterogenen Modellen und der globalen Wissensbasis. Der lokale Ontologieinterpreter führt Anfragen an die ihm zugeordnete lokale Ontologie aus und leistet damit unter anderem Übersetzungsdienste zwischen lokaler und globaler Begriffswelt.

- *Informationsermittler:* Wenn Informationen aus einem Modell benötigt werden, so tritt der Informationsermittler in Aktion. Er ist dafür zuständig, das ihm zugeordnete Modell auszulesen, die gewünschten Informationen zu ermitteln und dem Anfragenden zur Verfügung zu stellen. Er stellt bei einer Prüfung sozusagen die Schnittstelle zum Modell dar.

- *Modellintegrator:* Wird ein neues Modell erstellt bzw. dem System der Modelle hinzugefügt, so muss dieses in die Prüfung integriert werden. Dazu muss es einerseits in die Meta-

Meta-Ebene transformiert werden, damit es einheitlich betrachtet werden kann, und es muss beim Systemverwalter angemeldet werden.

- *Modellverwalter:* Die Verwaltung des Modells besteht aus der Aufgabe, in den einzelnen Modellelementen zu hinterlegen, welche Regeln Informationen über das entsprechende Element angefordert haben. So kann ermittelt werden, von welchen Regeln ein Modellelement betroffen ist. Dies ermöglicht es dem Benutzer wiederum, eine Regel für ein Modellelement oder das ganze Modell zu deaktivieren bzw. zu ignorieren.

- *Schnittstellenwächter:* Der Schnittstellenwächter wird benötigt, wenn die Konsistenz eines Modells permanent überwacht werden soll. Ist dies nicht der Fall, kann diese Rolle ignoriert werden. Der Schnittstellenwächter überwacht die Datenschnittstelle eines Engineering Werkzeugs und hat die Aufgabe, Änderungen im Modell zu erkennen und zu melden.

Mit Hilfe dieser Rollen werden im nächsten Unterkapitel die verschiedenen Agententypen identifiziert.

## 7.3 Agententypen

Bei der Identifikation der Agententypen gibt es zwei unterschiedliche Vorgehensweisen. Die Identifikation kann funktionsbasiert bzw. aufgabenbasiert oder entitätenbasiert durchgeführt werden [RSMG11]. Bei der aufgabenbasierten Dekomposition der Problemstellung stehen die zu erfüllenden Funktionen im Vordergrund und die Agententypen werden auf dieser Basis abgeleitet. Ein Vorteil bei dieser Vorgehensweise ist, dass die Aufgaben klar auf die Agententypen verteilt sind. Die aufwendigere Verwaltung der Entitäten und deren Daten kann sich jedoch nachteilig auswirken. Im Gegensatz zur aufgabenbasierten Dekomposition verknüpft die entitätenbasierte Dekomposition die Agententypen mit den im System vorhandenen Entitäten. Vorteile sind die klare Struktur, vor allem bei verteilten Systemen, und die mögliche Kapselung der Daten der jeweiligen Entitäten. Nachteile entstehen bei Aufgaben, die mehrere Entitäten betreffen und somit auf mehrere Agententypen verteilt werden müssen. In vielen Fällen ist es sinnvoll, beide Ansätze zur Dekomposition zu betrachten und die Agententypen nach einem hybriden Vorgehen zu identifizieren, um die Lösungsmöglichkeiten nicht einzuschränken.

In Abbildung 7.3 ist bereits eine Gruppierung der Rollen dargestellt, die die Aufgaben einerseits und die Entitäten andererseits berücksichtigt. Daraus lassen sich vier Agententypen ableiten, der *Koordinierungsagent*, der *Ontologieagent*, der *Regelagent* und der *Modellagent*. Da die Prüfung auf mehrere, heterogene Modellen angewandt werden soll, werden die Modelle jeweils durch einen Modellagenten repräsentiert. Dieser Agententyp entspricht also einer entitätenbasierten Betrachtungsweise. Das Gleiche gilt für den Ontologieagent, der die globale Ontologie repräsentiert, und die Regelagenten, die jeweils eine Regel darstellen. Der Koordinierungsagent kann

dem Benutzer zugeordnet werden. Seine Rolle als Prüfungskoordinator entspricht jedoch keiner Entität und der Agent würde somit einer aufgabenbasierten Dekomposition entspringen.

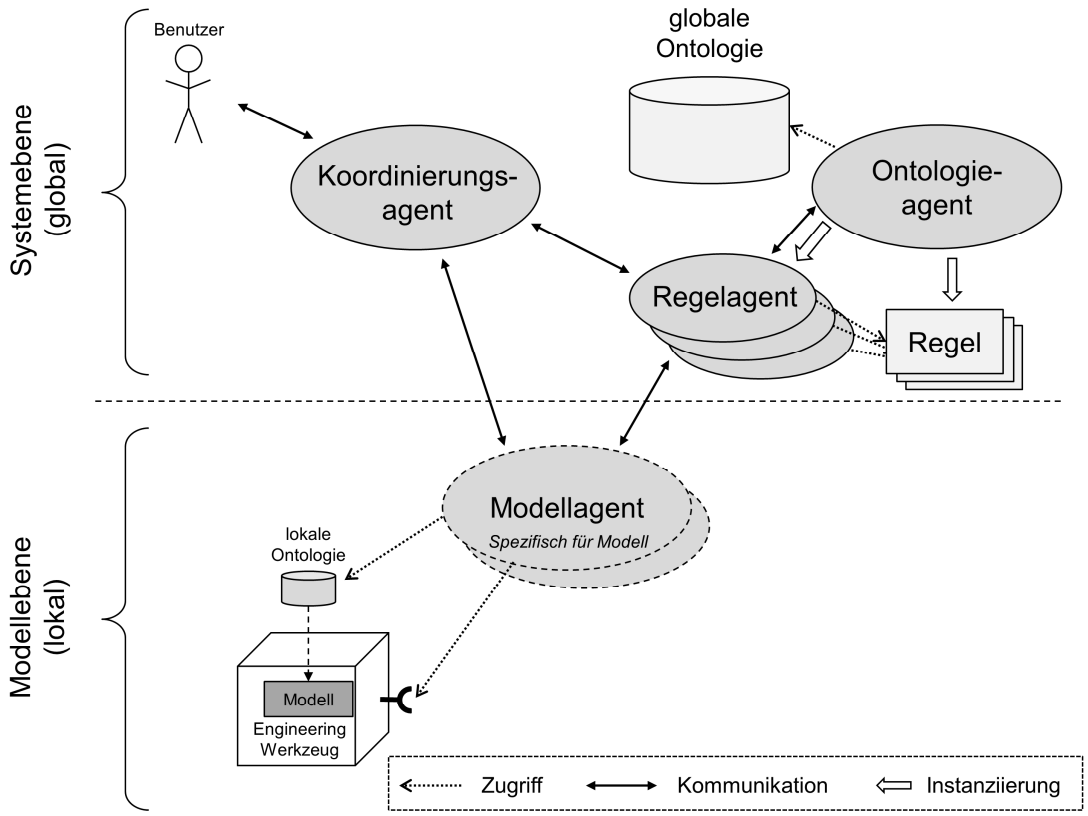


Abbildung 7.4: Agententypen im System

Abbildung 7.4 zeigt die identifizierten Agententypen in Verbindung mit den Entitäten im System. Vom Koordinierungsagent und vom Ontologieagent existiert jeweils nur eine Instanz, vom Regelagent und vom Modellagent befinden sich im Normalfall mehrere Instanzen im System. Der Modellagent ist in Abbildung 7.4 gestrichelt dargestellt, da er auf konzeptioneller Ebene zwar immer die gleiche Funktionalität aufweist, in der Realität jedoch auf Grund der unterschiedlichen Modelle und Engineering Werkzeuge respektive deren Schnittstellen nicht einheitlich implementiert werden kann.

### 7.3.1 Der Modellagent

Der Modellagent repräsentiert ein Modell im System der Modelle, weshalb für jedes Modell im System der Modelle ein Modellagent vorhanden sein muss. Er vereinigt die Rollen *Ontologieinterpreter (lokal)*, *Informationsermittler, Modellintegrator*, *Modellverwalter* und *Schnittstellenwächter*. Die beiden Rollen *Modellintegrator* und *Schnittstellenwächter* erfordern eine modell- und Engineering-Werkzeug-spezifische Implementierung für die Übersetzung des Modells in die Meta-Meta-Ebene und die Anbindung des Engineering Werkzeugs über dessen Datenschnittstelle. Deshalb wird der Modellagent als abstrakter Agententyp betrachtet (in den Abbil-

dungen gestrichelt dargestellt), der für das jeweilige Modell individuell implementiert werden muss [RaGö13b].

Er ist in der Lage, mit dem Koordinierungsagent und den Regelagenten zu kommunizieren. Außerdem hat er Zugriff auf das ihm zugeordnete Modell (über Datenschnittstelle) sowie dessen Beschreibung in Form einer lokalen Ontologie.

Im Folgenden wird der Aufbau dieses Agententyps an Hand der ihm zugeordneten Rollen definiert.

### Aufgaben als Ontologieinterpreter

Der Ontologieinterpreter muss Informationen aus der lokalen Ontologie bereitstellen. Er ist zudem in der Lage, (lokale) Begriffe aus dem Modell in globale Begriffe zu übersetzen und umgekehrt. Außerdem kann er die Struktur eines Modells aus der Ontologie auslesen. Der Ontologieinterpreter stellt somit die Schnittstelle des Modellagenten zur lokalen Ontologie dar und benötigt dementsprechend Zugriff auf diese.

### Aufgaben als Informationsermittler

Der Informationsermittler muss Anfragen von anderen Modellagenten beantworten und die gewünschten Informationen im lokalen Modell ermitteln. Dazu benötigt der Agent wiederum Zugriff auf das ihm zugeordnete Modell und muss in der Lage sein, die Anfragen zu interpretieren.

Der Modellagent benötigt also Zugriff auf das Modell. Dazu besitzt er eine Art Kopie des originalen Modells im Engineering Werkzeug, jedoch in Form des Meta-Meta-Modells (abstrahiertes Modell). So können die Modelle in allen Modellagenten einheitlich repräsentiert werden, auch wenn sie im Engineering Werkzeug gänzlich verschieden abgebildet sind. Zudem muss ein Anfrageinterpreter vorhanden sein, der die Informationsanfragen von anderen Agenten interpretieren und beantworten kann.

### Aufgaben als Modellintegrator

Um ein neues Modell in das System der Modelle zu integrieren, ist es notwendig, das Modell einzulesen und in die Meta-Meta-Form zu transferieren. Es muss also eine Schnittstelle, entweder die Datenschnittstelle eines Engineering Werkzeugs oder eine entsprechende Dateischnittstelle, und ein modellspezifischer Modelltransformator existieren. Nach der Transformation muss das neue Modell beim Systemverwalter angemeldet werden.

### Aufgaben als Modellverwalter

Der Modellverwalter benötigt Zugriff auf das vom Modellintegrator transformierte Modell in Meta-Meta-Form. Im Auftrag des Koordinierungsagenten bzw. des Benutzers muss er einzelne Regeln für manche Modellelemente sperren, damit diese nicht geprüft werden. Im Modell muss also die Möglichkeit vorgesehen werden, diese Informationen zu hinterlegen.

### Aufgaben als Schnittstellenwächter

Der Schnittstellenwächter prüft mit Hilfe der Schnittstelle zum Engineering Werkzeug permanent, ob vom Benutzer Änderungen am Modell durchgeführt wurden. Diese Funktionalität kann, abhängig von der Mächtigkeit der Schnittstelle des Engineering Werkzeugs, beispielsweise durch Beobachten der Nutzerinteraktionen oder durch zyklisch wiederholte Vergleiche des Modells mit einer internen Kopie des Modells erfolgen.

Nach diesen Analysen kann der Aufbau eines Modellagenten wie in Abbildung 7.5 abgebildet dargestellt werden.

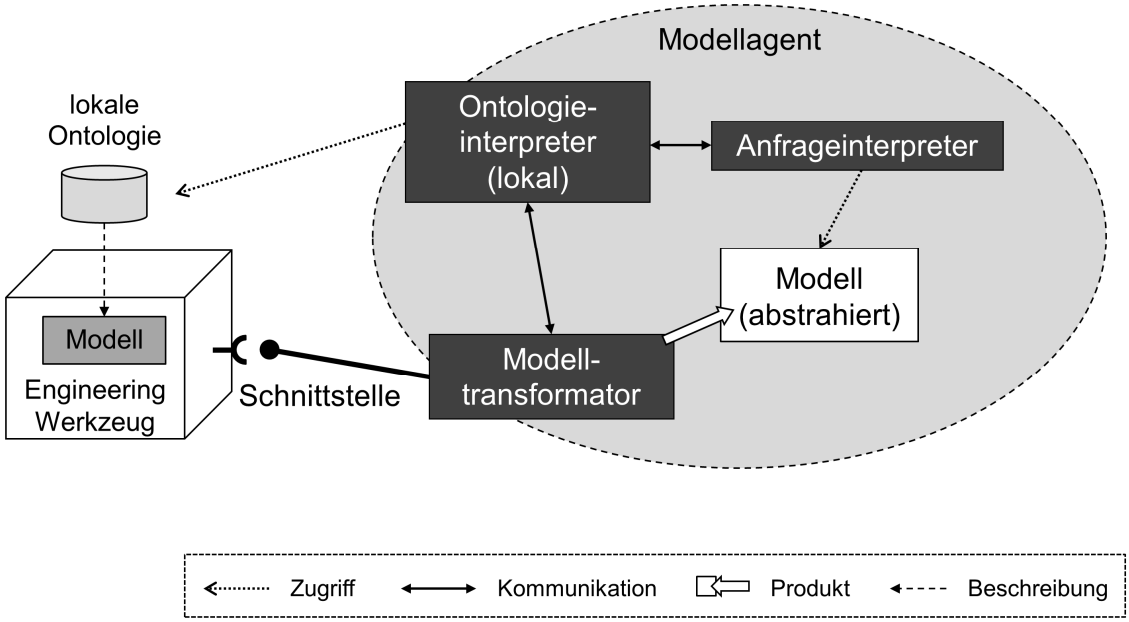


Abbildung 7.5: Aufbau eines Modellagenten

Mit Hilfe des Modelltransformators wird das abstrahierte, interne Modell erzeugt, indem das ursprüngliche Modell über die Schnittstelle ausgelesen wird. Dabei kommuniziert er mit dem Ontologieinterpreter, um mit dessen Hilfe die einzelnen Elemente des Modells von der lokalen in die globale Begriffswelt zu übersetzen. Der Anfrageinterpreter beantwortet Anfragen an den Inhalt des Modells und kann dabei ebenfalls auf den Ontologieinterpreter zurückgreifen.

## 7.3.2 Der Regelagent

Der Regelagent repräsentiert eine Regel im System zur Prüfung der Modelle, weshalb für jede Regel ein Regelagent vorhanden sein muss. Er übernimmt die Rolle *Regelinterpreter* und prüft die Modelle auf Einhaltung der ihm zugeordneten Regel.

Er ist in der Lage, mit allen anderen Agententypen im System zu kommunizieren. Vom Modellagenten erhält er Informationen aus dem jeweiligen Modell. Der Ontologieagent unterstützt ihn beispielsweise mit Berechnungsformeln für Größen, die zwar nicht direkt in einem Modell vor-

handen sind, mit den dort vorhandenen Größen jedoch berechnet werden können. Dem Koordinierungsagenten übermittelt er schließlich das Ergebnis seiner Prüfung.

Im Folgenden wird der Aufbau dieses Agententyps an Hand der ihm zugeordneten Rollen definiert.

### Aufgaben als Regelinterpreter

Der Regelinterpreter muss zunächst mit den Modellagenten kommunizieren, um zu ermitteln, welche konkreten Modellelemente mit der ihm zugeordneten Regel geprüft werden müssen. Dazu gehören auch Modellelemente, deren Typ vom eigentlichen Modellelementtyp, auf den die Regel angewendet werden muss, abgeleitet ist (Regel des Vaterelements wird auch auf Kindelemente angewendet). Anschließend fordert er die entsprechenden Informationen für die eigentliche Regelanwendung an und wertet die Regel aus. Er prüft also an Hand der Informationen aus den Modellen, ob die in der Regel geforderte Struktur vorhanden oder die Gleichheit von Werten etc. gegeben ist. Dabei müssen gegebenenfalls Formeln interpretiert und vorhandene Werte miteinander kombiniert werden. Die Ergebnisse werden registriert, um sie dem Koordinierungsagenten zu übermitteln. Wird eine zur Anwendung der Regel erforderliche Information in einem Modell nicht gefunden, so wird eine Anfrage an die Informationsbereitsteller der anderen Modelle des Systems (andere Modellagenten) gesendet. Kann die entsprechende Information bereitgestellt werden, so wird die Regel angewendet. Existiert die benötigte Information in keinem der Modelle (da evtl. kein Modell vorhanden ist, das den erforderlichen Inhalt enthält) oder wird der Regelagent von einem Modellagent zurückgewiesen (da der Benutzer die Regel für ein Modellelement deaktiviert hat), so kann die Regeln nicht angewendet werden. Dieses Ergebnis wird ebenfalls registriert, um später dem Benutzer mitgeteilt zu werden.

Nach diesen Analysen kann der Aufbau eines Regelagenten wie in Abbildung 7.6 abgebildet dargestellt werden.

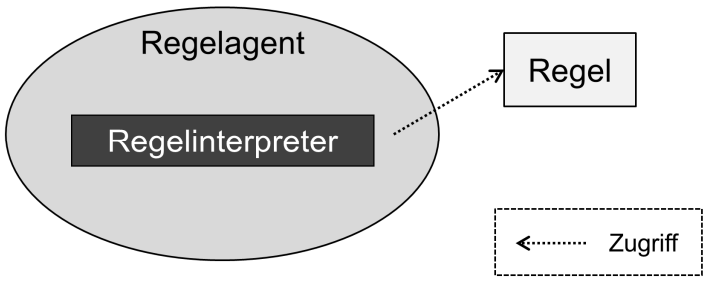


Abbildung 7.6: Aufbau eines Regelagenten

Der Regelinterpreter ist in der Lage die ihm zugeordnete Regel anzuwenden. Die dazu notwendigen Informationen erhält er durch Kommunikation mit den Modellagenten. Vom Ontologieagent erlangt er Informationen zur möglichen Kombination von verfügbaren Werten zu ge-

wünschten Werten (z.B. Formeln) oder Vererbungsbeziehungen zwischen Modellelementtypen. Ergebnisse werden von ihm zum Koordinierungsagent gemeldet.

### 7.3.3 Der Ontologieagent

Der Ontologieagent repräsentiert die globale Ontologie und existiert nur einmal im System. Er übernimmt die Rollen *Ontologieinterpreter* und *Regelinstanziierer*. Er initiiert selbst keine Kommunikation mit anderen Agenten, beantwortet jedoch Anfragen der Regelagenten.

Im Folgenden wird der Aufbau dieses Agententyps an Hand der ihm zugeordneten Rollen definiert.

#### Aufgaben als Ontologieinterpreter

Der Zugriff auf die globale Ontologie erfolgt über eine Schnittstelle. Diese muss verschiedene Funktionen bereitstellen. Sie muss in der Lage sein, Regeln aus der Ontologie abzuleiten. Dazu gehören nicht nur die explizit hinterlegten Regeln sondern es müssen auch die durch die Relationen zwischen den Begriffen implizit vorhandenen Regeln abgeleitet werden. Für jede Regel wird zudem ermittelt, für welche Elementtypen (z. B. „elektrischer Motor“) die Regel angewendet werden muss. Außerdem ermöglicht die Schnittstelle das Ermitteln von Formeln zur Berechnung von (Ziel-) Größen, indem die zur Berechnung verfügbaren Größen angegeben werden. Diese Formeln können von den Regelagenten angefragt werden, wenn die zur Auswertung einer Regel erforderlichen Informationen nicht in der richtigen Form in den Modellen vorliegen. Der Ontologieagent muss also auch mit den Regelagenten kommunizieren und deren Anfragen beantworten können.

#### Aufgaben als Regelinstanziierer

Der Regelinstanziierer erzeugt für jede vom Ontologieinterpreter ermittelte Regel eine Instanz. Anschließend instanziiert er einen zugehörigen Regelagenten, dem er die Regel und die Informationen zur Regel übergibt.

Nach diesen Analysen kann der Aufbau eines Ontologieagenten wie in Abbildung 7.7 abgebildet dargestellt werden.

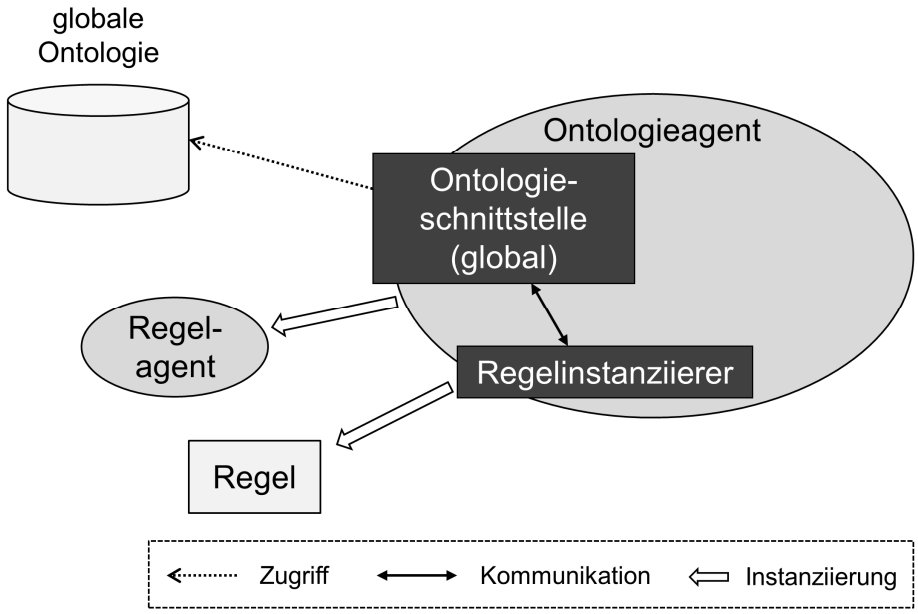


Abbildung 7.7: Aufbau eines Ontologieagenten

Der Ontologieagent besitzt also eine Schnittstelle zur globalen Ontologie und eine Funktion zur Instanziierung von Regeln und Regelagenten.

### 7.3.4 Der Koordinierungsagent

Der Koordinierungsagent übernimmt die Rollen *Systemverwalter*, *Prüfungskoordinator* und *Informationsbereitsteller*. Er existiert nur einmal und steht hierarchisch über den Modell- und Regelagenten. Auf Grundlage eines einzelnen Benutzers, der die Konsistenz des Systems der Modelle überwacht, übernimmt der Koordinierungsagent die Kommunikation mit diesem Benutzer bezüglich der Prüfung.

Wiederum werden die Aufgaben des Agenten betrachtet, um dessen Aufbau zu definieren.

#### Aufgaben als Systemverwalter

Der Systemverwalter besitzt den Überblick über das System der Modelle. Er weiß welche Modelle im System vorhanden sind, welchen Inhalt sie haben (z. B. physikalische Struktur, logische Struktur oder Verhalten) und welcher Modellagent für das jeweilige Modell zuständig ist. Dazu besitzt er eine entsprechende Liste der Modelle, die jedoch der Einfachheit halber an den Directory Facilitator der Agentenplattform (Gelbe Seiten) ausgelagert werden kann.

#### Aufgaben als Prüfungskoordinator

Eine Prüfung der Modelle wird vom Prüfungskoordinator initiiert und überwacht. Auf Wunsch des Benutzers werden die Regelagenten beauftragt, alle oder nur eine Auswahl der Modelle auf Einhaltung ihrer jeweiligen Regel zu prüfen. Wurden vom Benutzer einzelne Regel deaktiviert, so werden die betreffenden Regelagenten nicht beauftragt. Die Ergebnisse werden zentral ge-

sammelt und der Koordinator erklärt die Prüfung für beendet, wenn alle Ergebnisse eingetroffen sind. Es muss deshalb eine Liste der Ergebnisse vorhanden sein. Ebenso muss eine Liste der Regeln sowie der zugehörigen Regelagenten existieren.

### Aufgaben als Informationsbereitsteller

Die Ergebnisse einer Prüfung werden dem Benutzer vom Informationsbereitsteller präsentiert. Die Ergebnisse enthalten neben dem Endergebnis auch die angewendeten, ignorierten und deaktivierten Regeln sowie bei Regelverletzungen die Informationen aus den Modellen, die zur Verletzung geführt haben. Diese werden aufbereitet und dem Benutzer bereitgestellt. Neben der Logik zur Aufbereitung der Informationen ist für den Koordinierungsagent eine graphische Benutzungsoberfläche (GUI) notwendig.

Der resultierende Aufbau des Koordinierungsagenten ist in Abbildung 7.8 dargestellt.

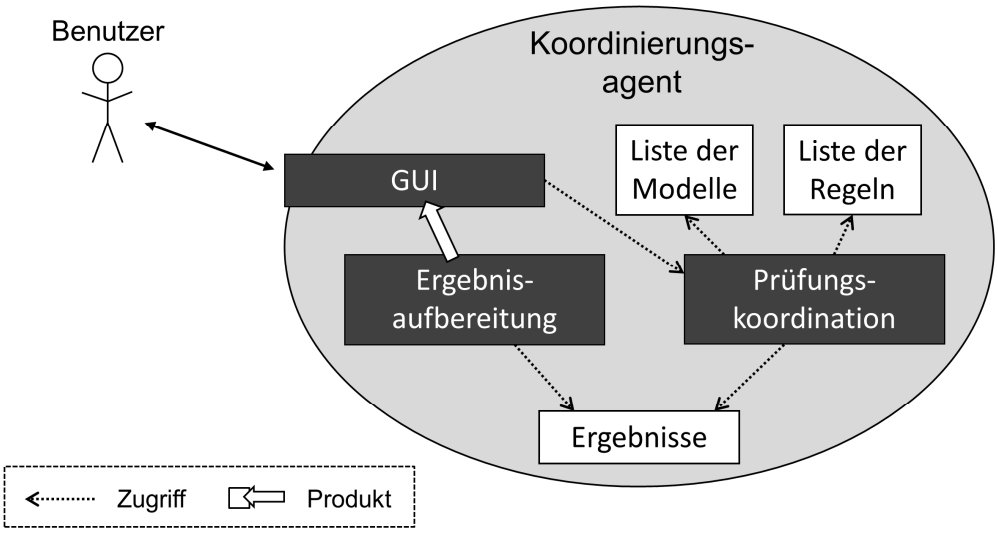


Abbildung 7.8: Aufbau des Koordinierungsagenten

Der Benutzer ist in der Lage, über die GUI einen Prüfungsprozess anzustoßen und erhält die aufbereiteten Ergebnisse zurück. Die für einen Benutzer nicht sichtbaren Interaktionen zwischen den Agenten werden im folgenden Unterkapitel definiert.

## 7.4 Interaktionen der Agenten

Im System existieren vier Agententypen, zwischen denen Interaktionen stattfinden können. Ein graphischer Überblick über die notwendigen Kommunikationskanäle ist in Abbildung 7.9 dargestellt.

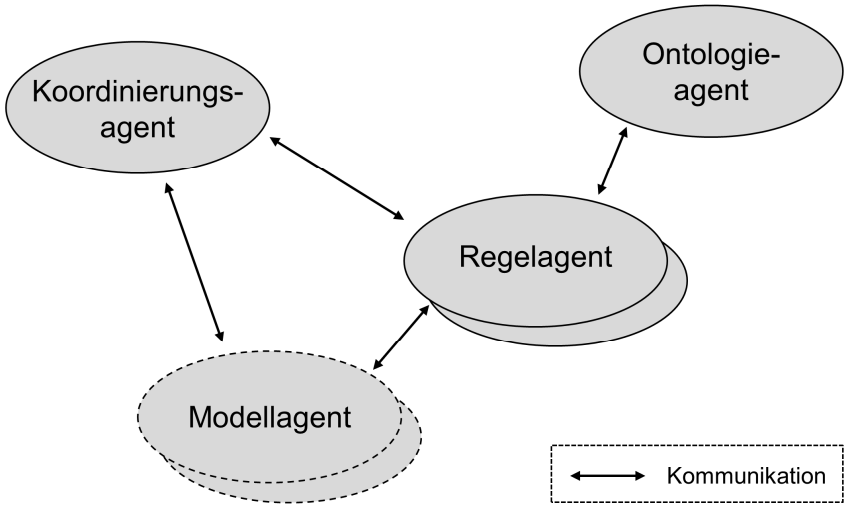


Abbildung 7.9: Kommunikationskanäle zwischen den Agenten

### 7.4.1 Interaktionen zwischen Koordinierungsagent und Regelagenten

Zu Beginn einer Prüfung beauftragt der Koordinierungsagent die Regelagenten, die Modelle auf Einhaltung ihrer jeweiligen Regel zu prüfen (vgl. Abbildung 7.10). Die entsprechende Nachricht, die dazu gesendet wird, enthält eine Liste der Modelle, die geprüft werden sollen. So ist es möglich, einzelne Modelle auf Benutzerwunsch von der Prüfung auszuschließen. Die Regelagenten bestätigen den Erhalt des Prüfungsauftrags und führen die Prüfung durch. Sobald ein Regelagent seine Prüfung abschließt, sendet er die Ergebnisse an den Koordinierungsagenten. Die Ergebnismeldung enthält die Ergebnisse jeder Anwendung der Regel (positiv oder negativ geprüft, ignoriert, deaktiviert), auf welche Modellelemente die Regel angewendet wurde und die in die jeweiligen Bedingungen eingesetzten Daten aus den Modellen.

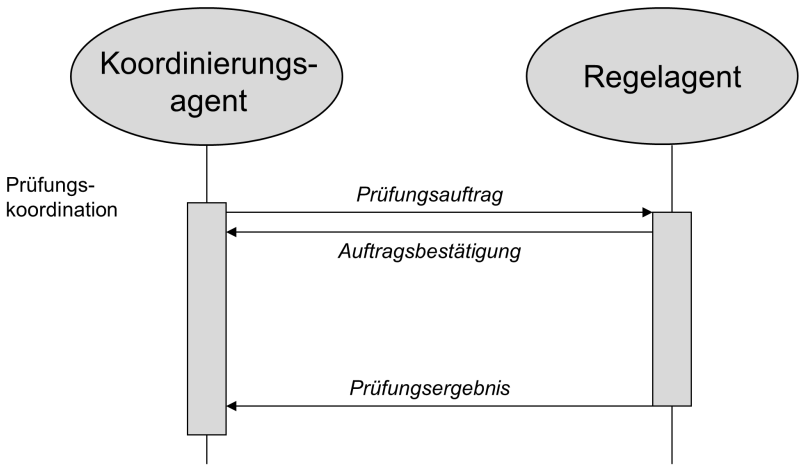


Abbildung 7.10: Sequenzdiagramm der Interaktion zwischen Koordinierungsagent und Regelagent

### 7.4.2 Interaktionen zwischen Koordinierungsagent und Modellagenten

Wird dem Prüfungssystem ein neues Modell hinzugefügt, so muss der Modellagent, der das Modell im System repräsentiert, dieses beim Koordinierungsagent anmelden. Dazu sendet er dem Koordinierungsagent eine Nachricht, die die Bezeichnung und den Typ des Modells enthält (vgl. Abbildung 7.11). Je nach Realisierung und Einsatz des vorliegenden Konzepts können weitere Informationen wie der Autor oder der Ort des Modells sinnvoll sein. Der Koordinierungsagent bestätigt anschließend die Aufnahme des Modells in das Prüfungssystem.

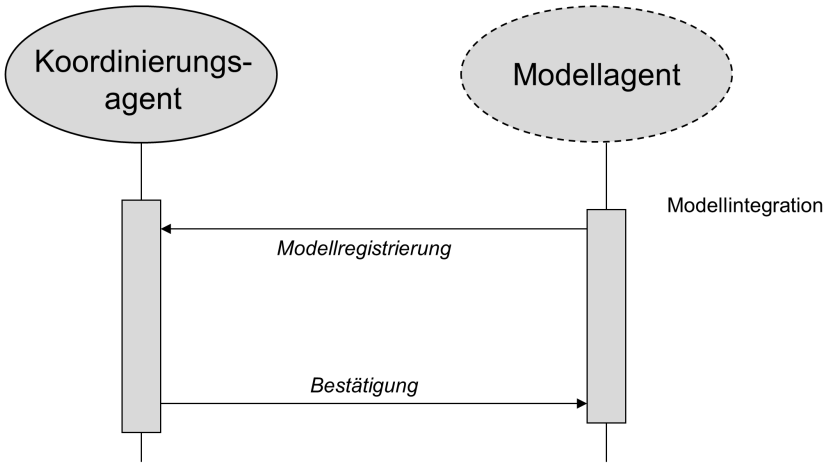


Abbildung 7.11: Sequenzdiagramm der Interaktion zwischen Koordinierungsagent und Modellagent

Sofern im Modellagenten die Rolle des Schnittstellenwächters umgesetzt ist und das Modell permanent auf Änderungen überwacht wird, wird der Koordinierungsagent über Änderungen im Modell in Kenntnis gesetzt. Dieser bestätigt den Erhalt der Information mit einer entsprechenden Nachricht.

Ein weiterer Nachrichtenaustausch zwischen diesen beiden Agententypen wird vom Koordinierungsagenten initiiert. Er informiert den Modellagenten, welche Modellelemente für welche Regeln geblockt werden sollen (teilweise Deaktivierung von Regeln durch Benutzer). Diese Nachricht wird vom Modellagenten ebenfalls quittiert.

### 7.4.3 Interaktionen zwischen Regelagenten und Modellagenten

Die Regelagenten haben die Aufgabe zu prüfen, ob die ihnen zugeordnete Regel von den Modellen erfüllt wird. Dazu benötigen sie Informationen aus den Modellen, die sie von den Modellagenten anfordern (siehe Abbildung 7.12).

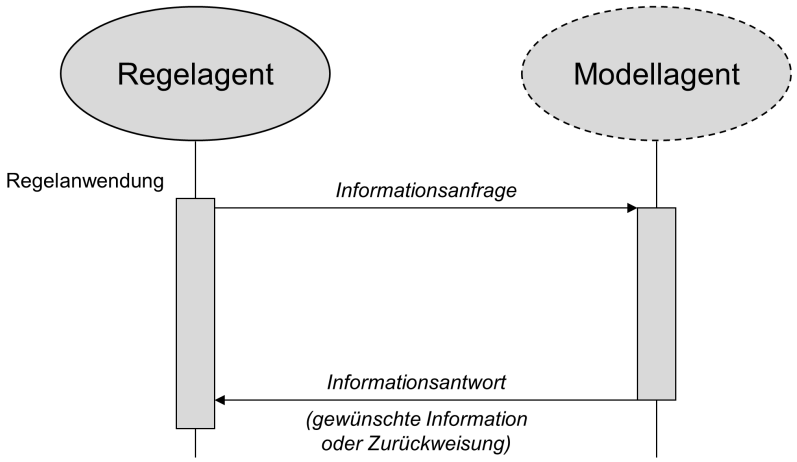


Abbildung 7.12: Sequenzdiagramm der Interaktion zwischen Regelagent und Modellagent

Die Informationsanfrage kann entweder die Anfrage nach einem bestimmten Wert, beispielsweise einem Zahlenwert, oder nach der Existenz eines Modellelements sein. In beiden Fällen muss die Anfrage alle notwendigen Informationen enthalten, damit diese beantwortet werden kann. So muss bei der Anfrage nach einem Wert die jeweilige Quelle (z. B. Typ des Modellelements) mitgeliefert werden. Die Interpretation der Anfrage und damit die Ermittlung der Antwort ist Aufgabe des Modellagenten, der die jeweilige Anfrage erhalten hat.

Die Antwort auf eine Informationsanfrage enthält, sofern im Modell vorhanden, die gewünschte Information. Ist die angeforderte Information im Modell nicht enthalten, so wird die Anfrage abgewiesen.

### 7.4.4 Interaktionen zwischen Regelagenten und dem Ontologieagent

Regelagenten bekommen vom Ontologieagent nach ihrer Instanziierung ihre jeweilige Regel zugeordnet. Sie müssen jedoch in der Lage sein, auch während eines Prüfungsvorgangs mit dem Ontologieagenten zu kommunizieren. Wenn der Regelagent beispielsweise zur Ermittlung der Einhaltung seiner Regel eine Größe benötigt, deren Wert nicht im Modell vorhanden ist, so benötigt er eine Formel aus der Ontologie, um die erforderliche Größe mit Hilfe anderer, vorhandener Größen zu berechnen. Eine solche Formel fragt er beim Ontologieagent an, der sie ihm zur Verfügung stellt. Der entsprechende Nachrichtenaustausch ist in Abbildung 7.13 dargestellt.

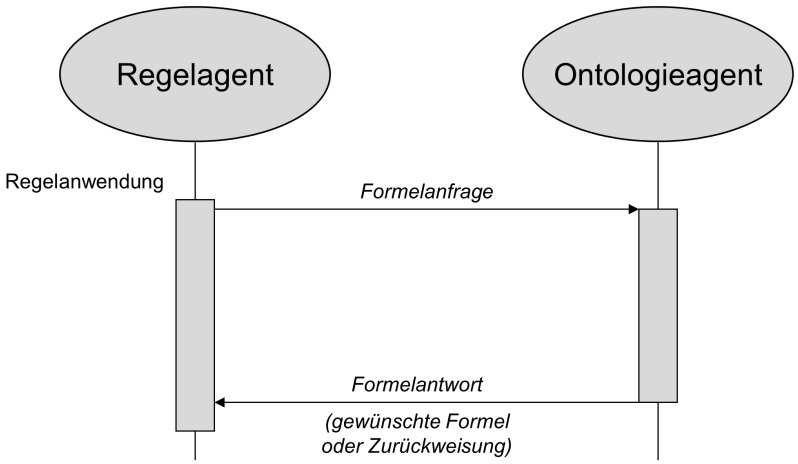


Abbildung 7.13: Sequenzdiagramm der Interaktion zwischen Regelagent und Ontologieagent

Ähnlich verhält es sich mit Anfragen zu Vererbungsbeziehungen zwischen Begriffen. Muss beispielsweise die Existenz eines „Motors“ überprüft werden, das Modell enthält jedoch einen „Elektrischen Motor“, so stellt der Regelagent beim Ontologieagent die Anfrage, ob der „Elektrische Motor“ ein Unterbegriff von „Motor“ ist.

Entscheidend für eine sinnvolle und erfolgreiche Kommunikation zwischen allen Agenten ist die Tatsache, dass sich alle Nachrichten in der globalen Begriffswelt bewegen und keine lokalen Begriffe enthalten. Nur dann ist ein Agent in der Lage, eine Nachricht zu interpretieren.

In diesem Kapitel wurde das agentenbasierte Konzept zur Prüfung von heterogenen Modellen vorgestellt. Nach einem Überblick über die Entitäten im System wurden die Rollen für die Agenten definiert. Anschließend wurden daraus vier Agententypen abgeleitet. Die Modellagenten repräsentieren jeweils ein Modell im System und stellen die darin modellierten Informationen bereit. Dazu besitzen sie Schnittstellen zum Engineering Werkzeug, das das Modell enthält, und zur jeweiligen lokalen Ontologie. Die Regelagenten stellen jeweils eine Regel zur Prüfung der Modelle dar, deren Einhaltung sie innerhalb und zwischen den Modellen prüfen. Der Ontologieagent repräsentiert die globale Ontologie, leitet daraus Regeln ab und stellt Informationen aus der Ontologie bereit. Der Koordinierungsagent überwacht die Prüfung, sammelt die Ergebnisse und stellt die Schnittstelle zum Benutzer dar. Die Interaktionen zwischen den einzelnen Agenten dienen der Koordinierung der Prüfung, dem Erlangen von Informationen oder der Übermittlung von Prüfungsergebnissen. Die zugehörigen Nachrichten sind in der globalen Begriffswelt formuliert. Diese ist in der globalen Ontologie abgelegt. Deren Aufbau sowie der Aufbau der lokalen Ontologien wird im folgenden Kapitel vorgestellt.

# 8 Konzept zur ontologiebasierten Wissensrepräsentation

Das Wissen, das für eine Prüfung der Modelle benötigt wird, muss wie die Modelle in formaler und in einer für Softwaresysteme verarbeitbaren Form vorliegen. Dies soll mit Hilfe von Ontologien erreicht werden. Zunächst wird eine Übersicht über das notwendige Wissen geschaffen. Anschließend wird vorgestellt, wie die Struktur und der Inhalt eines Modells beschrieben werden können. Der Aufbau einer lokalen Ontologie zeigt, wie diese Informationen in der lokalen, einer einem Modell zugeordneten Ontologie abgelegt sind. Die zur Prüfung erforderlichen Regeln werden ebenfalls vorgestellt und es wird ausgeführt, wie diese gehandhabt werden. Sie sind in der globalen Ontologie abgelegt, die die globale Begriffswelt repräsentiert. Zum besseren Verständnis folgt zunächst jedoch ein Überblick über das Wissen und die Ziele, die mit der Wissensrepräsentation erreicht werden sollen.

## 8.1 Übersicht und Ziel der Wissensrepräsentation

Das für eine Konsistenzprüfung erforderliche Wissen lässt sich grundsätzlich in zwei Bereiche aufteilen. Der erste Bereich enthält das Fachwissen bezüglich der Domäne, in der ein Entwicklungsprojekt durchgeführt wird. Dieses wird im Folgenden als modellneutrales Fachwissen bezeichnet. Aus ihm können später die Zusammenhänge zwischen den Modellen und damit die Regeln zu deren Prüfung abgeleitet werden. Der zweite Bereich des Wissens ist modellspezifisch und enthält die Beschreibung eines Modells. Diese ermöglicht die Interpretation eines Modells auf der Ebene des modellneutralen Fachwissens.

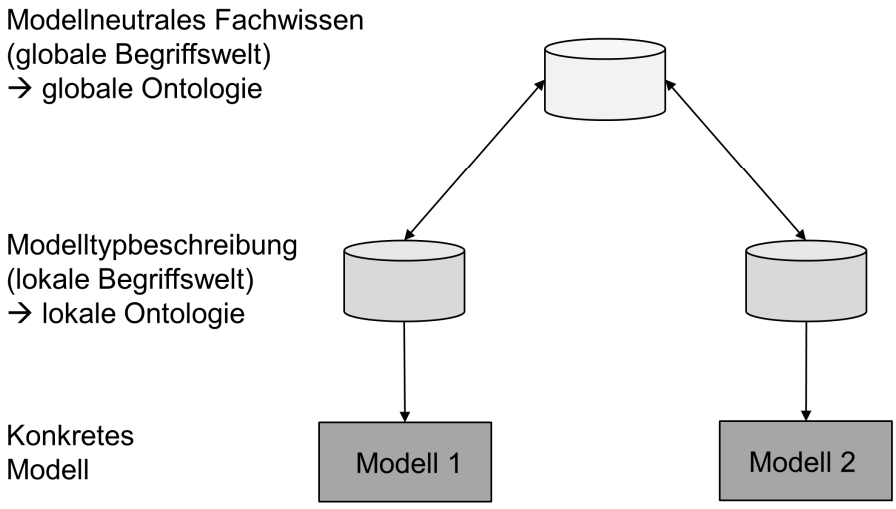


Abbildung 8.1: Globale und lokale Begriffswelten

Die beiden Wissensbereiche müssen in einer rechnerverarbeitbaren Form abgebildet werden, um die Prüfung der Modelle automatisiert durchführen zu können. Dazu werden die in Kapitel 4 vorgestellten Ontologien eingesetzt und das Wissen wird entsprechend formalisiert. In Abbildung 8.1 werden globale und die lokalen Ontologien dargestellt, die die globale und die lokalen Begriffswelten abbilden.

In den folgenden Unterkapiteln wird erarbeitet, welchen Inhalt die globale und die lokalen Begriffswelten besitzen müssen und wie diese strukturiert sind.

## 8.2 Beschreibung von Modellen

In Kapitel 6.4 wurde gezeigt, wie ein Meta-Meta-Modell aussehen muss, mit dessen Hilfe nahezu beliebige Modelle abgebildet werden können. Es wurde allerdings auch erwähnt, dass das eindeutige Verständnis eines Modells existentiell für eine Modellprüfung ist. Damit muss eine Beschreibung eines Modells existieren, die dieses Verständnis ermöglicht. Diese Beschreibung muss alle Informationen enthalten, die notwendig sind, um den Inhalt bzw. die Aussagen des Modells auf die übergeordnete, modellneutrale Wissensebene zu transferieren. Für die Modelle der UML existieren beispielsweise Spezifikationen, die unter anderem definieren, wie ein UML-Modell aufgebaut ist und welche Elemente es enthält [OMG11]. Auch für andere Modelle bzw. deren Modellierungssprachen existieren entsprechende Beschreibungen. Ohne diese Beschreibungen wäre es nicht möglich, die Modelle sinnvoll zu benutzen, da zum Beispiel die Bedeutung einzelner Modellelemente nicht definiert wäre. In den folgenden Abschnitten wird daher untersucht, welches Wissen über ein Modell vorhanden sein muss, um dieses zu benutzen bzw. eindeutig zu interpretieren und zu verändern (bezogen auf die automatisierte Prüfung der Modelle), und wie dieses Wissen dargestellt werden kann.

### 8.2.1 Beschreibung der Struktur

Zur Benutzung und zur Prüfung eines Modells müssen zunächst dessen strukturelle Informationen bekannt sein. Dazu gehören die Elemente (Elementtypen), die das Modell enthalten kann, und die Regeln, die festlegen, wie diese Elemente verknüpft werden dürfen. In der Sprachlehre wird die Strukturbeschreibung als Syntax bezeichnet. Die entsprechenden Informationen sind im Meta-Modell (M2) vorhanden. Die Symbolik, die beschreibt, wie die Elemente des Modells dargestellt werden, ist zwar auch Bestandteil des Meta-Modells, wird für die Prüfung jedoch nicht benötigt. Sie ist nur notwendig, wenn ein Modell für einen Benutzer dargestellt werden soll.

Die Beschreibung der strukturellen Informationen muss formal und unabhängig vom Meta-Modell bzw. dessen Modellierungssprache erfolgen. Da das Modell später in eine rechnerverarbeitbare Form übertragen werden soll, eignet sich beispielsweise ein UML-Klassendiagramm, um die Struktur eines Modells bzw. dessen Modellelementen darzustellen. Innerhalb diesem

werden alle Elemente als Klassen definiert, verwandtschaftliche Beziehungen (Vererbung) können ebenfalls einfach abgebildet werden. Abbildung 8.2 zeigt einen Ausschnitt aus dem Klassendiagramm der Wirkstruktur.

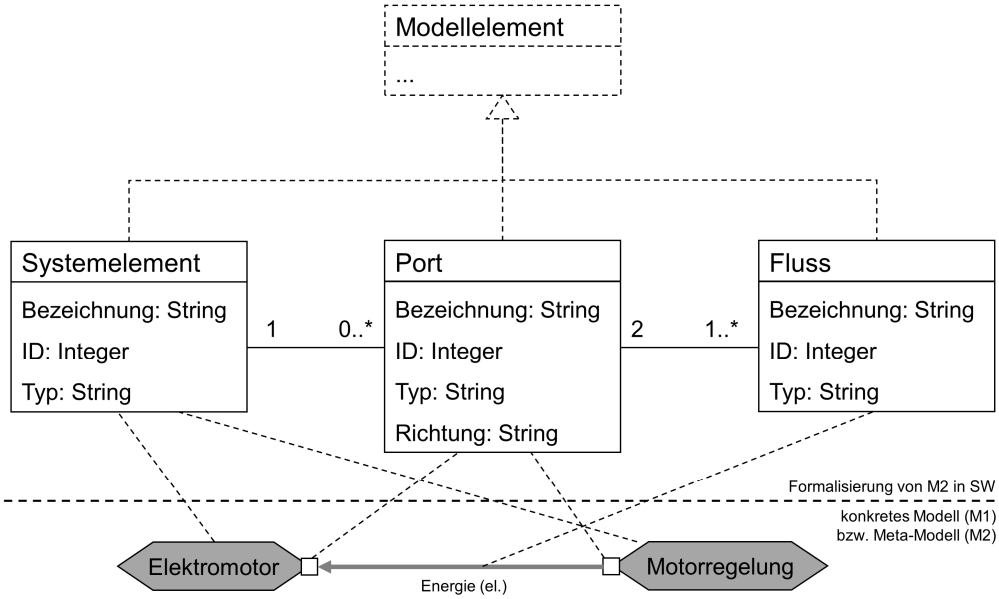


Abbildung 8.2: Ausschnitt des Klassendiagramms der Wirkstruktur

Dabei werden Systemelemente, Ports und Flüsse jeweils als eigene Klasse abgebildet. Die Assoziationen zwischen den Klassen beschreiben, wie die jeweiligen Elemente verknüpft werden können. Kardinalitäten geben an, wie viele Elemente jeweils verbunden werden dürfen. Kompositionen werden zur Vereinfachung in Form von Kardinalitäten ausgedrückt. Das so entstandene Klassendiagramm beschreibt das Meta-Modell formal und eindeutig und kann so in die Beschreibung eines Modells integriert werden. Bei der Prüfung kann das Klassendiagramm ausgelesen und mit dem realen Modell verglichen werden.

## 8.2.2 Beschreibung des Inhalts

Nachdem die syntaktische Beschreibung definiert wurde, muss nun die semantische Beschreibung folgen. Die Struktur der Elemente eines Modells ist bekannt und es muss definiert werden, was die einzelnen Elemente bedeuten. Dazu wird jedes Element mit Hilfe allgemeiner Begriffe beschrieben. Diese Begriffe stammen aus der globalen Begriffswelt, die das gesamte Gebiet, in diesem Fall der Mechatronik, beschreibt. Die globalen Begriffe werden in einer globalen, das heißt allgemeingültigen Ontologie (globale Ontologie bzw. Begriffswelt) definiert, deren Aufbau in Kapitel 8.5 erläutert wird. Hier wird zunächst davon ausgegangen, dass diese Ontologie bzw. die darin enthaltenen Begriffsdefinitionen existieren.

Alle Elemente eines Modells bzw. die Elementtypen werden mit Begriffen der globalen Ontologie verknüpft, damit deren Bedeutung mit Hilfe der Ontologie abgeleitet werden kann. Abbildung 8.3 zeigt ein Beispiel hierzu.

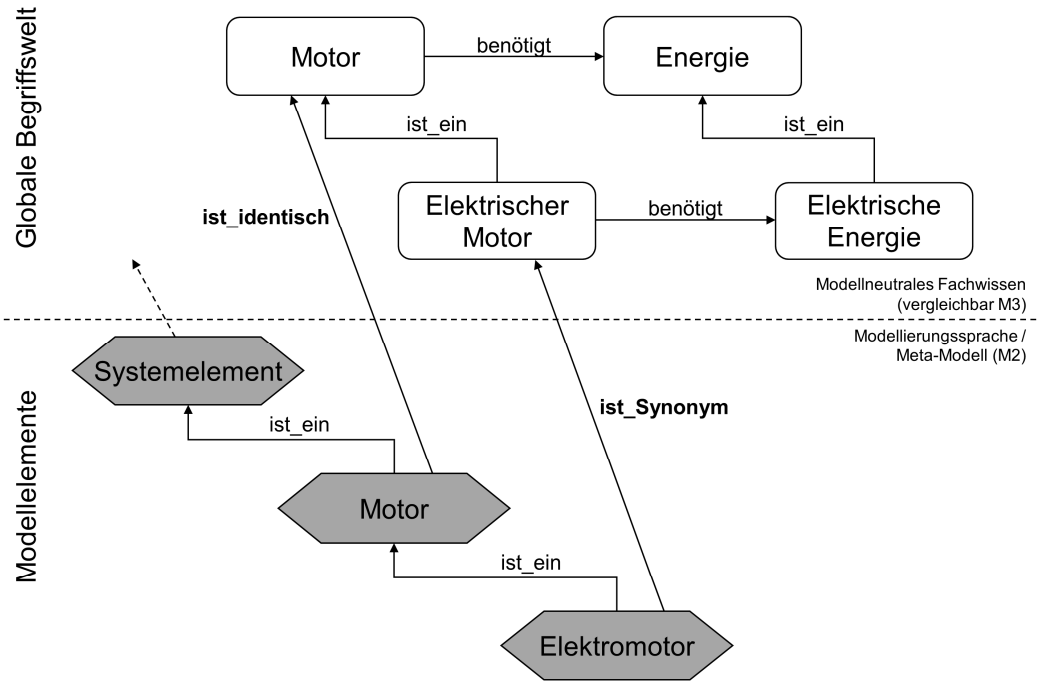


Abbildung 8.3: Beschreibung von Modellelementen bzw. lokalen Begriffen mit Hilfe von globalen Begriffen

Das Systemelement „Motor“ (lokal) ist identisch mit dem globalen Begriff „Motor“. Die Spezialisierung „Elektromotor“ ist zwar prinzipiell identisch mit dem globalen Begriff „Elektrischer Motor“, jedoch lautet der Begriff anders. Deshalb bildet der lokale Begriff „Elektromotor“ ein Synonym zum globalen Begriff. Zur besseren Unterscheidung wurden in der Abbildung für die Systemelemente die Symbole des Modells „Wirkstruktur“ benutzt.

Ebenso werden alle weiteren Begriffe, die im Modell zum Beispiel als Eigenschaftsbezeichnung, -typ oder -wert verwendet werden, mit Hilfe von globalen Begriffen dargestellt.

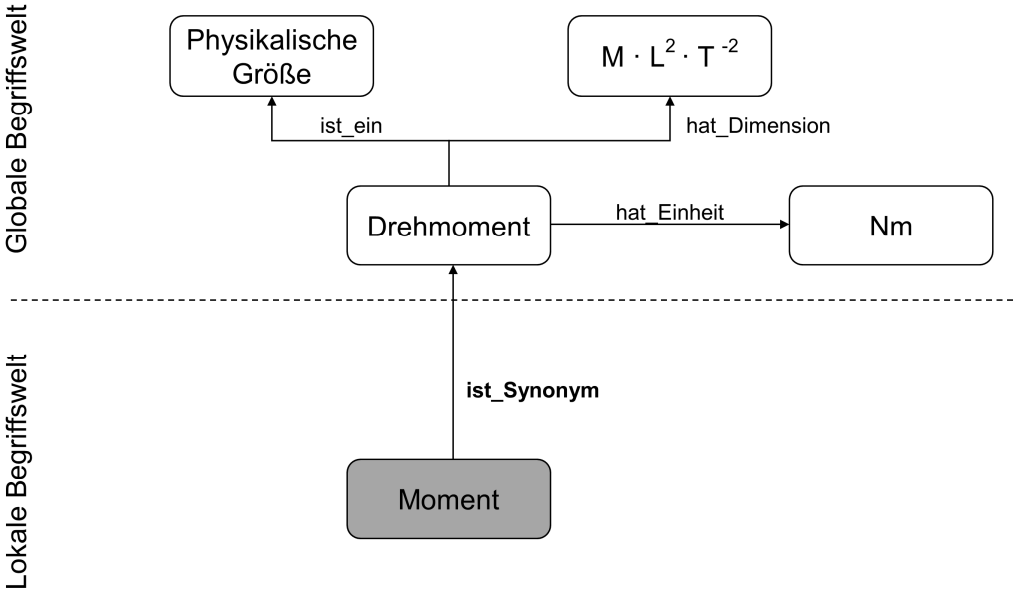


Abbildung 8.4: Verknüpfung von lokalen und globalen Begriffen

Wie in Abbildung 8.4 dargestellt ist, wird der lokale Begriff „Moment“ mit dem globalen Begriff „Drehmoment“ verknüpft. Mit Hilfe dieser Verknüpfung kann bei einer Analyse des Modells beim Auftauchen des Begriffs „Moment“ darauf geschlossen werden, dass dieser gleichbedeutend mit dem globalen und damit bekannten Begriff „Drehmoment“ ist. So kann ein Modell auch dann interpretiert werden, wenn die lokale von der globalen Begriffswelt abweicht.

Schließlich müssen lokal verwendete Einheiten für physikalische Größen in globale Einheiten umgerechnet werden können. Die globalen Einheiten werden mit Hilfe der SI-Einheiten ausgedrückt. Für lokale Einheiten muss also definiert werden, welche Größe sie beschreiben und wie die Umrechnung in SI-Einheiten erfolgen muss. Abbildung 8.5 zeigt beispielhaft die Verknüpfung der lokalen Einheit „PS“ für die Leistung mit der globalen Einheit „W“. Einerseits ist ersichtlich, dass die lokale Größe die Leistung (globaler Begriff) beschreibt, andererseits ist der Umrechnungsfaktor zur SI-Einheit Watt angegeben.

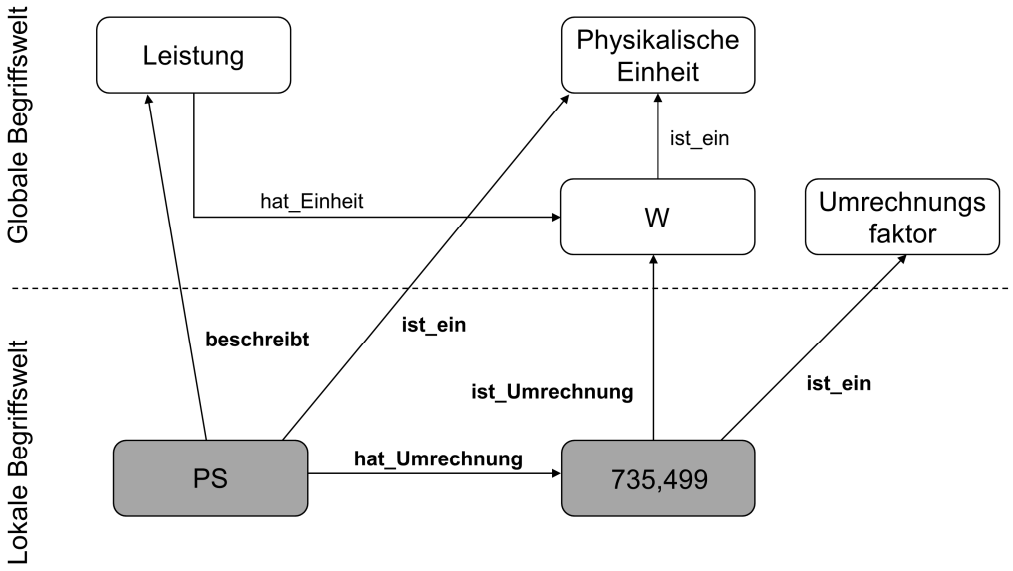


Abbildung 8.5: Beschreibung einer physikalischen Größe mit Nicht-SI-Einheit

Auf diese Weise können alle inhaltlichen Informationen eines Modells interpretiert werden, da deren Bedeutung mittels der Verknüpfungen zur globalen Begriffswelt erfasst werden kann.

Die Informationen bzw. Daten eines konkreten Modells (Modell von System „xy“) sind nicht Bestandteil der Beschreibung eines Modells. Diese ist allgemein und projektübergreifend gefasst und ist mit dem Meta-Modell (M2) vergleichbar. Die Modellbeschreibung geht jedoch über das Meta-Modell hinaus, da dieses nur strukturelle Information enthält, und die Modellbeschreibung zusätzlich auch semantische Informationen zur Interpretation des Modellinhalts umfasst. Die projektspezifischen Informationen in einem konkreten Modell müssen dagegen lediglich in einem allgemeinen, modellunspezifischen Zusammenhang abgebildet werden können, was durch die Formalisierung im Kapitel 6.4 erreicht wurde. Abbildung 8.6 zeigt die Formalisierung und die Beschreibung eines Modells in einem gemeinsamen Überblick.

In der unteren Hälfte ist die Abbildung des Inhalts eines Modells in einem allgemeinen Informationscontainer, einer Instanz des Meta-Meta-Modells, dargestellt. Dies ermöglicht, jedes beliebige Modell einheitlich und neutral darzustellen. In der oberen Hälfte werden die Informationen zur Beschreibung des Modells formal und damit für ein verarbeitendes System lesbar ausgedrückt. Das Modell ist im Meta-Modell (Beschreibungssprache) definiert und strukturelle Informationen sowie Informationen den Inhalt betreffend werden getrennt abgelegt. Die Struktur (Syntax) wird in Form eines Klassendiagramms, der Inhalt (Bedeutung/Semantik) wird mittels Verknüpfungen zu globalen Begriffen definiert. So kann die Beschreibung zur Interpretation eines Modells für jedes Modell bereitgestellt werden, sofern die globale Begriffswelt ausreichend groß ist, um alle lokalen, modellspezifischen Begriffe zu beschreiben.

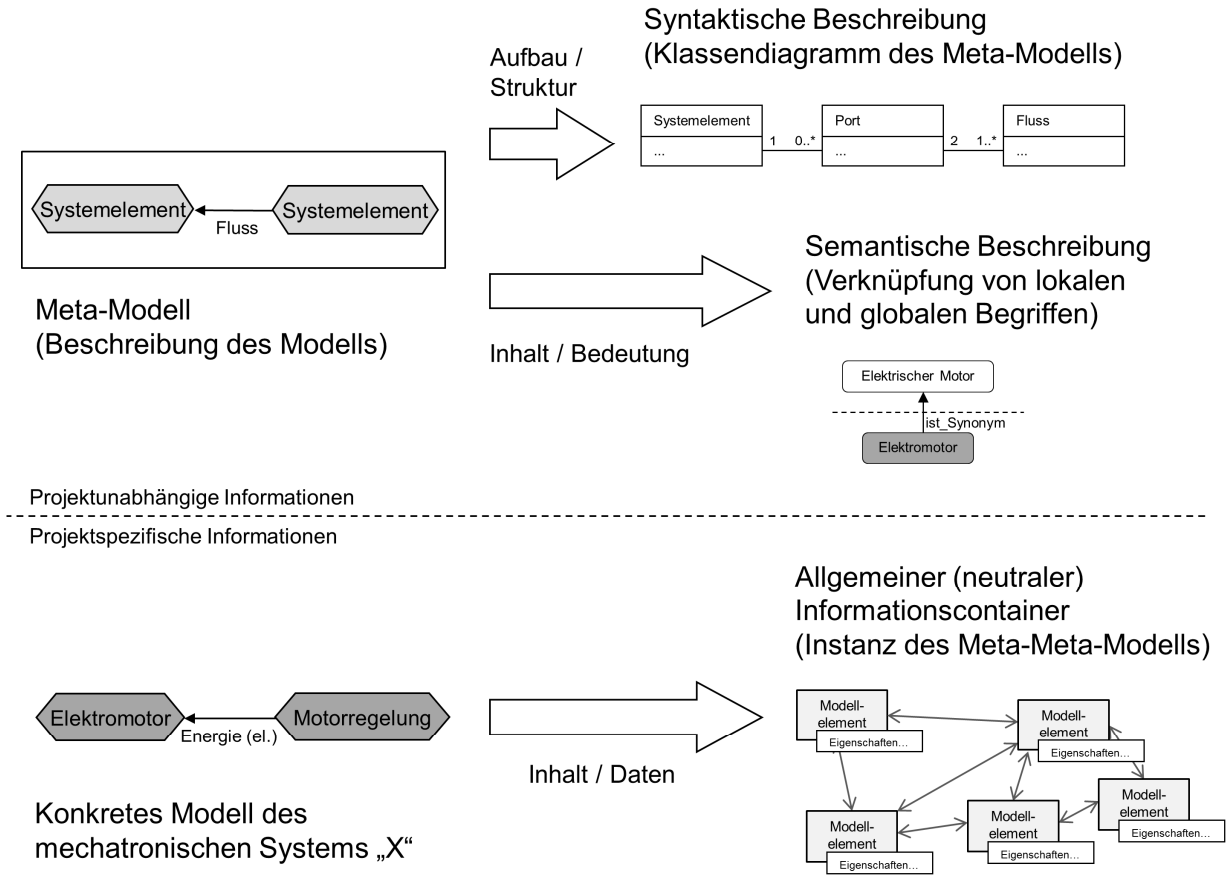


Abbildung 8.6: Überblick über die Formalisierung und Beschreibung eines Modells

## 8.3 Aufbau einer lokalen Ontologie

Die im vorherigen Unterkapitel vorgestellte Modellbeschreibung muss mit dem jeweiligen Modell bereitgestellt werden. Dazu wird diese formal dargestellt und in Form einer Ontologie dem Modell zugeordnet. Der Aufbau dieser als „lokal“ bezeichnete Ontologie wird im Folgenden dargelegt.

Auch wenn hier von einer einzigen lokalen Ontologie gesprochen wird, wird der Umstand ausgenutzt, dass sich verschiedene Ontologien verknüpfen lassen. So besteht die lokale Ontologie aus mehreren Teilontologien, die jeweils einen Aspekt der Modellbeschreibung enthalten. Dies ermöglicht eine bessere Strukturierung und damit eine einfachere Erstellung und Pflege der lokalen Ontologien.

Die Syntax eines Modells wird wie im vorherigen Unterkapitel dargestellt als UML-Klassendiagramm beschrieben. Dieses wird schließlich in der lokalen Ontologie abgebildet. Abbildung 8.7 zeigt das UML-Klassendiagramm der Wirkstruktur (aus Abbildung 8.2). Der hervorgehobene Ausschnitt dient als Beispiel der in Abbildung 8.8 abgebildeten Repräsentation dieses UML-Klassendiagramms in der Ontologie.

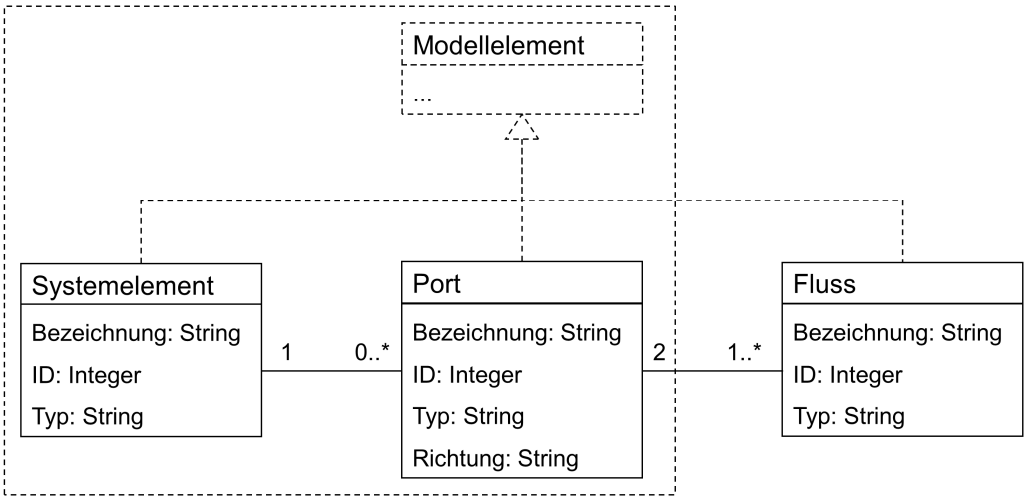


Abbildung 8.7: Abgebildeter Ausschnitt des UML-Klassendiagramms

Die obere Hälfte von Abbildung 8.9 zeigt die Begriffs- oder Klassenebene der Ontologie und definiert das Meta-Modell des benötigten Ausschnitts eines UML-Klassendiagramms. Sie ist für alle Modellbeschreibungen identisch und kann damit auch in der globalen Ontologie abgelegt werden. Die untere Hälfte zeigt die Instanzebene und repräsentiert das UML-Klassendiagramm der Wirkstruktur. Dieser Teil ist modellspezifisch und ist damit dem Modell zugeordnet. Zur besseren Lesbarkeit sind die wichtigen Informationen in den Namen der Instanzen hinterlegt und diese sind in Anführungszeichen gesetzt. So ist zu erkennen, dass die Instanz „Systemelement“ des Begriffs „Klasse“ von der Instanz „Modellelement“, ebenfalls Instanz von „Klasse“, erbt. Die Assoziation „I456“ verknüpft die Klassen „Systemelement“ und „Port“ mit den entsprechenden Kardinalitäten. Die Kardinalitäten auf der Begriffsebene sind in die Relationen der Ontologie integriert, da diese global und damit statisch sind. Die Kardinalitäten des UML-Klassendiagramms auf Instanzebene sind explizit modelliert, da sie modellspezifisch sind.

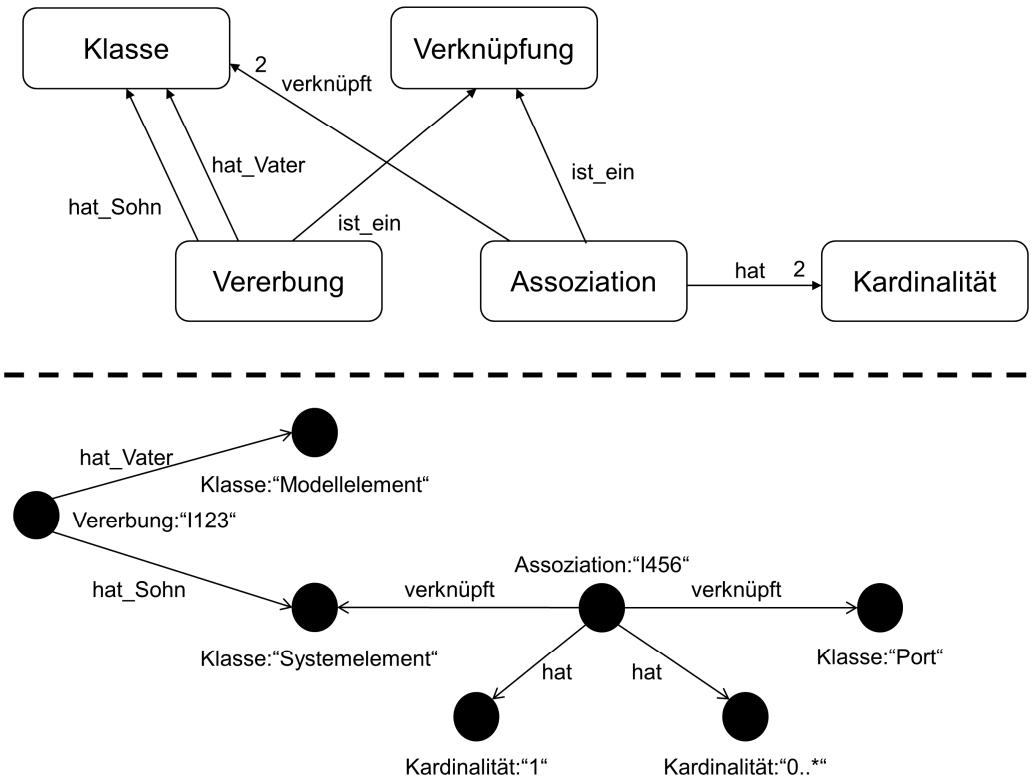


Abbildung 8.8: Ontologische Repräsentation des UML-Klassendiagramms der Wirkstruktur aus Abbildung 8.2 (Ausschnitt)

Die Semantik eines Modells wurde bereits im vorherigen Unterkapitel dargestellt. Hier soll nur noch einmal darauf hingewiesen werden, dass es je nach Modell, das beschrieben wird, sinnvoll ist, die Semantik (Begriffsverknüpfungen von lokalen und globalen Begriffen) auf mehrere Teilontologien zu verteilen.

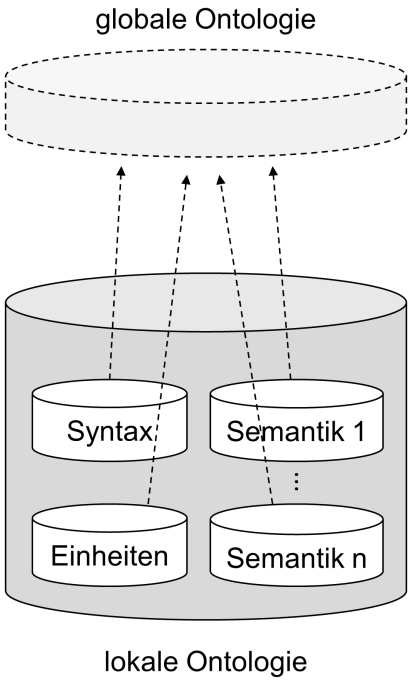


Abbildung 8.9: Struktur einer lokalen Ontologie

Eine lokale Ontologie setzt sich damit wie in Abbildung 8.9 dargestellt zusammen. Neben der Strukturbeschreibung, die ein UML-Klassendiagramm darstellt, existieren eine oder mehrere Teilontologien, die die Semantik enthalten. Diese müssen die Bedeutung der Modellelemente beinhalten und alle im Modell möglicherweise verwendeten Begriffe müssen mit Hilfe globaler Begriffe definiert sein. Die lokale Ontologie darf keine eigenständigen Begriffe (Stammbegriffe) enthalten, die nicht auf Begriffe der globalen Ontologie zurückzuführen sind. Ansonsten ist keine Abstraktion der Begriffe in die globale Begriffswelt möglich. Werden im Modell Einheiten für Größen verwendet, die in der globalen Ontologie nicht definiert sind, muss die Umrechnung dieser Einheiten ebenfalls enthalten sein. Nur wenn alle Aussagen (lokale Begriffe, Einheiten, etc.) mit Hilfe der globalen Ontologie beschrieben werden können, können diese mit Hilfe der in der globalen Begriffswelt beschriebenen Regeln geprüft werden.

## 8.4 Beschreibung von Regeln

Mit Hilfe der Formalisierung der Modelle und den Modellbeschreibungen ist es jetzt möglich, beliebige Modelle rechnergestützt zu verarbeiten. Im nächsten Schritt ist es notwendig, die Modelle auf Inkonsistenzen zu prüfen. Dabei ist zwischen Inkonsistenzen in der Struktur (z. B. nicht zulässige Verknüpfungen zwischen Modellelementen) und Inkonsistenzen im Inhalt (z. B. falsche Attributwerte) zu unterscheiden. Fehler in der Struktur betreffen immer nur ein einzelnes Modell, inhaltliche Fehler können innerhalb eines Modells oder zwischen mehreren Modellen auftreten. Damit eine Prüfung der Konsistenz automatisiert durchgeführt werden kann, ist die Definition von Regeln notwendig. Diese können von einem Rechnersystem angewendet werden und durch deren Verletzung können Inkonsistenzen erkannt werden. Um die Verwaltung der Regeln einfacher und deren Anwendung weniger fehleranfällig zu gestalten, werden alle Regeln im Positivfall formuliert. Das heißt, wenn eine Regel erfüllt ist, ist die Konsistenz bezüglich dieser Regel gegeben. Wenn eine Regel nicht erfüllt ist, so liegt ein Problem (eine Inkonsistenz) vor und dieses muss behoben werden. Wenn alle Regeln erfüllt sind, so kann, sofern die Regeln vollständig sind, von einem konsistenten Modell bzw. einem konsistenten System von Modellen ausgegangen werden (vgl. Anforderung 5: Inkonsistenzen erkennen).

Der Aufbau der Regeln ist angelehnt an [Wagn08] und sieht, wie in [RaGö12a] vorgestellt, ein Regelobjekt vor, welches eine Vorbedingung und eine Hauptbedingung besitzt. Die entsprechende Struktur ist in Abbildung 8.10 dargestellt und wird in den folgenden Abschnitten erweitert werden.

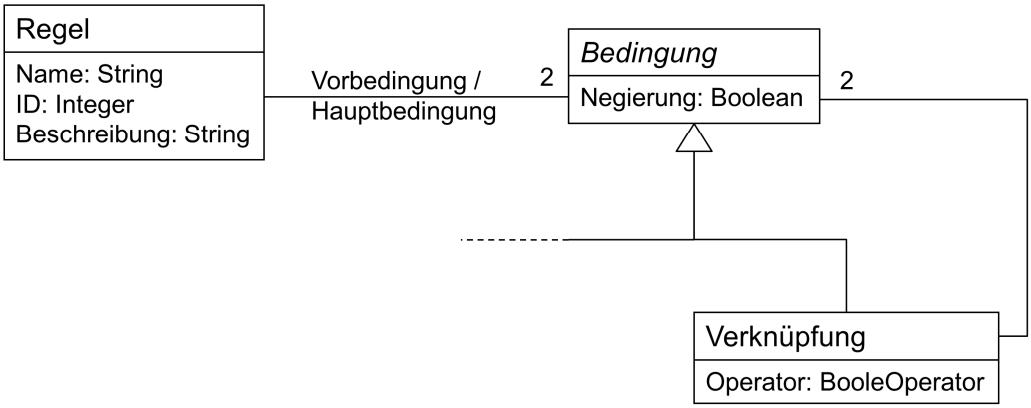


Abbildung 8.10: Aufbau einer Regel

Die Regel besitzt einen *Namen*, eine *ID* und einen *Beschreibung*. Diese Eigenschaften dienen hauptsächlich der Verwaltung der Regeln. Zudem besitzt eine Regel zwei Bedingungen. Mit Hilfe der Vorbedingung wird entschieden, ob eine Regel zur Anwendung kommt. In vielen Fällen wird diese Vorbedingung immer wahr sein, das heißt die Regel wird immer angewendet werden. Es sind jedoch auch Fälle vorstellbar, in denen Regeln nur bei bestimmten Konstellationen, beispielsweise bei der Existenz von elektrischen Spannungen über 230 Volt, zum Einsatz kommen. Diese Zweiteilung der Bedingungen ermöglicht einerseits eine Geschwindigkeitsoptimierung der Prüfung (unterstützt Anforderung 2: zeitliche Flexibilität), da nicht immer alle Regeln vollständig überprüft werden müssen, andererseits wird der Problemlösungsvorgang vereinfacht, da die eigentliche Aussage einer Regel getrennt von der Einsatznotwendigkeit betrachtet werden kann.

Die Bedingungen wiederum sind so gestaltet, dass bei deren Auswertung nur „wahr“ oder „nicht wahr“ als Resultat zurückgegeben werden. Damit können Bedingungen einfach negiert (über entsprechendes Attribut) oder zu komplexeren Bedingungen verknüpft werden, wobei ein binärer Baum entsteht. Als Operatoren werden die Booleschen Operatoren *UND*, *ODER* und *XODER* verwendet. Durch die Möglichkeit der Negierung lassen sich so alle logischen Operationen erstellen.

Bedingungen besitzen neben den verknüpften Bedingungen weitere Untertypen, die im Folgenden ermittelt und definiert werden. Zunächst ist eine Bedingung notwendig, die immer erfüllt ist, um beispielsweise die Vorbedingung einer Regel auf „wahr“ zu setzen. Für die Ermittlung von weiteren Typen muss der Einsatzzweck der Regeln genauer untersucht werden.

Wie bereits bei der Beschreibung eines Modells wird auch bei den Regeln zwischen Regeln, die die Struktur eines Modells betreffen, und Regeln, die den Inhalt eines Modells betreffen, unterschieden. Das zur Prüfung erforderliche Wissen bzw. die Regeln müssen unabhängig von den zu prüfenden Modellen sein, um Erweiterungen beim Hinzufügen eines neuen Modelltyps zu vermeiden. Für inhaltliche Regeln ist dies gegeben, sofern der neue Modelltyp keine Aspekte modelliert, die in der globalen Wissensbasis nicht berücksichtigt sind. Die Struktur eines Modells

ist jedoch modelltypspezifisch und kann nicht in der globalen Wissensbasis hinterlegt sein. Ansonsten müsste diese bei jedem Aufkommen eines neuen Modelltyps erweitert werden. Regeln, die die Struktur betreffen, müssen also aus der Beschreibung eines Modells abgeleitet werden.

### 8.4.1 Strukturelle Regeln

Die Beschreibung eines Modells enthält Informationen über dessen Struktur in Form eines Klassendiagramms. In diesem Klassendiagramm sind alle Modellelementtypen sowie die erlaubten Verknüpfungen zwischen diesen enthalten. Daraus muss mittels eines Algorithmus ein Regelsatz für die strukturelle Prüfung abgeleitet werden. Das Modell selbst kann als Objektstruktur verstanden werden, die dem Klassendiagramm entsprechen muss.

Um die unterschiedlichen Bedingungstypen im Regelsatz zu ermitteln, muss zunächst untersucht werden, welche Sachverhalte überprüft werden müssen, um die Modellierung gemäß dem Klassendiagramm bestätigen zu können. Folgende Auflistung enthält die wichtigsten Sachverhalte in Form von Fragen, mit deren Hilfe im weiteren Verlauf die Bedingungstypen abgeleitet werden.

- Besitzt die Klasse (das Modellelement) alle erforderlichen Attribute (Eigenschaften)?
- Existieren nur erlaubte Assoziationen zwischen den Modellelementen und werden die Kardinalitäten eingehalten?
- Ist die Vererbung berücksichtigt?

Zur Überprüfung der Eigenschaften wird ein Bedingungstyp benötigt, der die Existenz eines Objekts, in diesem Fall einer Eigenschaft wiedergeben kann. Dazu müssen die Eigenschaften eines Modellelements nach einem gewissen Filterkriterium, hier der Name der Eigenschaft, ausgewählt werden und die Anzahl der Ergebnisse muss eins sein. Es wird also ein Bedingungstyp benötigt, der Zahlenvergleiche beschreiben kann. Zudem ist es erforderlich, Objekte nach verschiedenen Kriterien filtern und zählen zu können. Die verschiedenen Bedingungstypen sind in Abbildung 8.11 dargestellt.

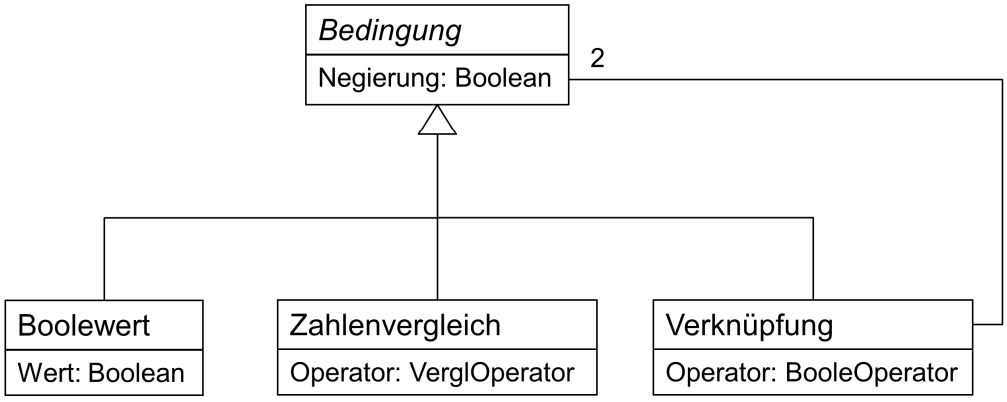


Abbildung 8.11: Bedingungstypen für die strukturellen Regeln

Abbildung 8.12 zeigt den Ausschnitt des Bedingungstyps *Zahlenvergleich*. Dabei kann ein Zahlenwert entweder einen festgelegten Wert (aus dem Modell bzw. dem Klassendiagramm abgeleitet) oder die Anzahl von Objekten einer bestimmten Gruppe repräsentieren. Im letzteren Fall existiert eine Quelle, die beschreibt, woher der Wert kommt bzw. von welcher Gruppe die Objekte gezählt werden.

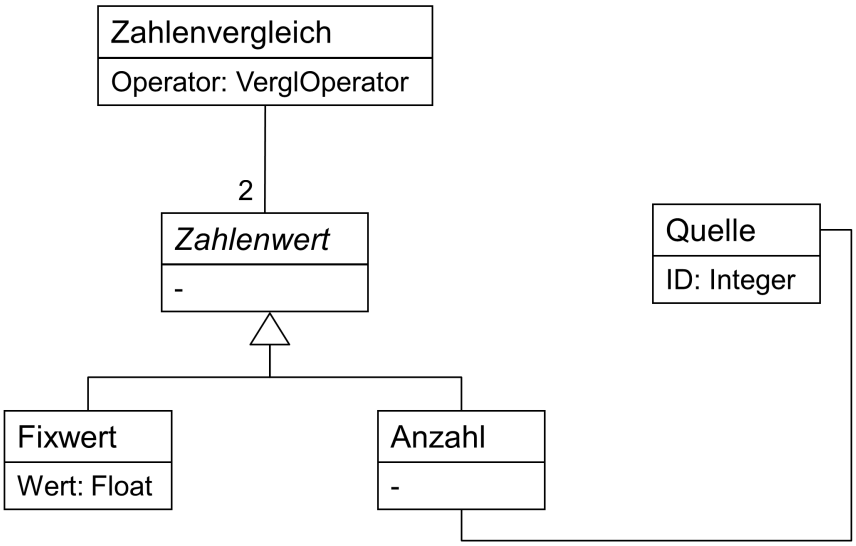


Abbildung 8.12: Ausschnitt des Bedingungstyps Zahlenvergleich

Die erlaubten Assoziationen werden geprüft, indem die Anzahl aller Verknüpfungen eines Objekts mit der Anzahl der Verknüpfungen zu erlaubten anderen Objekte verglichen wird. Sind beide Anzahlen nicht identisch, so muss eine nicht erlaubte Verknüpfung existieren. Für diese Überprüfung sind dieselben Bedingungstypen wie beim vorherigen Beispiel erforderlich. Es muss jedoch zusätzlich möglich sein, Filterkriterien zu kombinieren. Die Kardinalitäten können mit einem Vergleich der erlaubten Anzahl an Verknüpfungen zu einem Objekttyp mit der ermittelten Anzahl geprüft werden.

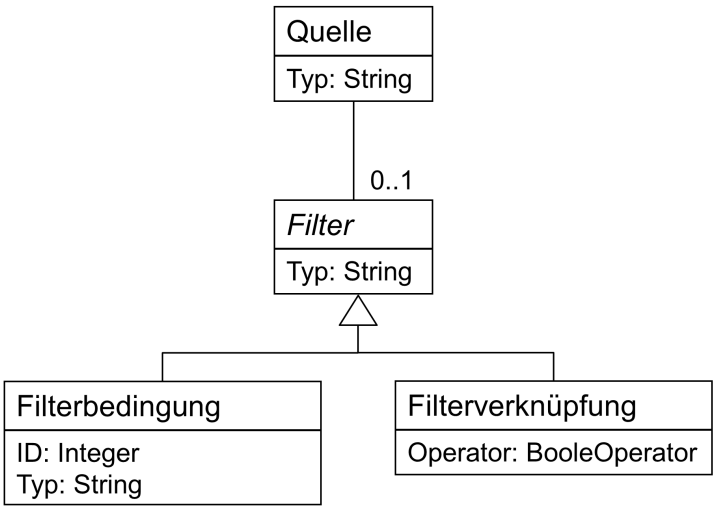


Abbildung 8.13: Quelle mit Filtermöglichkeit

Abbildung 8.13 zeigt die Quelle eines Werts, die mit einem Filter versehen werden kann. Dieser besitzt Filterbedingungen, die wiederum mit Booleschen Operatoren verknüpft werden können. Die Vererbung wird berücksichtigt, indem Regeln neben der Klasse, für die sie definiert sind, auch für alle Unterklassen angewendet werden.

Die für die strukturelle Prüfung erforderlichen Regeln werden automatisiert aus den Modellbeschreibungen abgeleitet. Das dort enthaltene Klassendiagramm, welches die Struktur eines Modells definiert, enthält alle dazu notwendigen Informationen. Eine manuelle Pflege dieser Regeln ist also nicht notwendig.

### 8.4.2 Inhaltliche Regeln

Auf Grund der Heterogenität der Modelle, müssen die inhaltlichen Regeln abstrakt und modellneutral sein, damit sie auf beliebige Modelle angewendet werden können. Sie dürfen sich also nur auf Begriffe der globalen Begriffswelt beziehen. Der Aufbau der Regeln erfolgt analog zu den formalen Regeln, jedoch sind zusätzliche Bedingungstypen erforderlich und die Auswertung der Regeln muss flexibler erfolgen. Prinzipiell sind dabei beliebige Bedingungstypen möglich, sofern die zugehörige Auswertungsfunktionalität im Softwaresystem vorhanden ist.

Im weiteren Verlauf werden an Hand von verschiedenen Beispielen einige Erweiterungen der Bedingungstypen abgeleitet und es wird auf die Besonderheiten bei der Auswertung eingegangen. Die Beispiele müssen sowohl Regeln, die innerhalb eines Modells gelten, als auch Regeln, die modellübergreifend angewendet werden müssen (abhängig von den verwendeten Modellen), enthalten. Zudem sollen Regeln unterschiedlicher Komplexität herangezogen werden, um möglichst viele Bedingungstypen abzubilden. Eine vollständige Abbildung aller Bedingungstypen ist auf Grund der Offenheit für neue Modelle nicht möglich. Die folgende Auflistung stellt die herangezogenen Beispiele vor:

- Eine Information muss eine Quelle und eine Senke besitzen. Eine Senke kann beispielsweise ein Zustandsübergang in einem das Verhalten beschreibenden Modell sein, eine Quelle eine Lichtschranke in einem die Systemelemente enthaltenden Modell. Quelle und Senke können jedoch auch beide im selben Modell vorkommen, wie die Stellgröße als Information zwischen Geschwindigkeits- und Motorregelung. Diese Regel kann also sowohl modellintern als auch modellübergreifend zur Anwendung kommen.

- Der Energieerhaltungssatz muss gelten, das heißt bei der Wandlung von zum Beispiel elektrischer in mechanische Leistung (elektrischer Motor) kann nicht mehr mechanische Leistung entstehen als elektrische zugeführt wird. Sofern der Wirkungsgrad bekannt ist, müssen zusätzlich Verluste berücksichtigt werden. Zudem muss gelten, dass die Summe des Leistungsbedarfs der Verbraucher nicht größer als die mögliche Leistungsbereitstellung der Quelle ist. Dies gilt sowohl für die mechanische als auch für elektrische Leistung.

- Die Anforderungen müssen eingehalten werden, was so viel bedeutet, dass alle Modelle konsistent zu dem die Anforderungen beschreibenden Modell sein müssen. Anforderungen können beispielsweise die maximalen Abmessungen, den Energieverbrauch, den gewünschten Materialdurchsatz oder Schutz gegen Umgebungseinflüsse beinhalten.

- Manche Systemelemente benötigen andere, um funktionsfähig zu sein. So ist für den Betrieb eines elektrischen Motors eine elektrische Energiequelle mit motorabhängigen Leistungsanforderungen erforderlich. Soll der Motor mit variablen Drehzahlen betrieben werden, so wird bei Synchron- oder Asynchronmotoren zusätzlich ein Umrichter, bei Gleichstrommotoren ein Stromregler benötigt. Ist der genaue Motortyp nicht bekannt, so ist allgemein eine Motorregelung erforderlich.

Das erste Beispiel ist mittels dreier Regeln realisierbar. Für das als Informationssenke fungierende Modellobjekt muss gelten, dass genau ein weiteres Objekt existiert, das dieselbe Information (Stringvergleich der Bezeichnung) aussendet. Dasselbe gilt für die Informationsquelle, indem mindestens ein weiteres Objekt existieren muss, das die ausgesendete Information annimmt. So sind beide Richtungen abgedeckt. Zudem muss gelten, dass eine Verbindung zwischen Quelle und Senke existiert. Hierbei muss bei der späteren Auswertung erschwerend berücksichtigt werden, dass keine direkte Verknüpfung beider Modellobjekte notwendig ist, sondern auch mehrere andere Objekte dazwischen existieren können. Dies ist beispielsweise der Fall, wenn ein Modell explizit Schnittstellen modelliert, wie es bei der erweiterten Wirkstruktur der Fall ist. Es ist lediglich erforderlich, dass die Information von der Quelle bis zur Senke übertragen werden kann, also im vorliegenden Beispiel ein Pfad entsprechender Informationsflüsse vorhanden ist. Abbildung 8.14 zeigt den Ausschnitt aus einem Objektdiagramm, in dem die für dieses Beispiel wichtigen Aussagen markiert sind. Zum besseren Verständnis zeigt die Abbildung das Beispiel nicht als abstraktes Meta-Meta-Modell sondern benutzt die Darstellung der Partialmodelle.

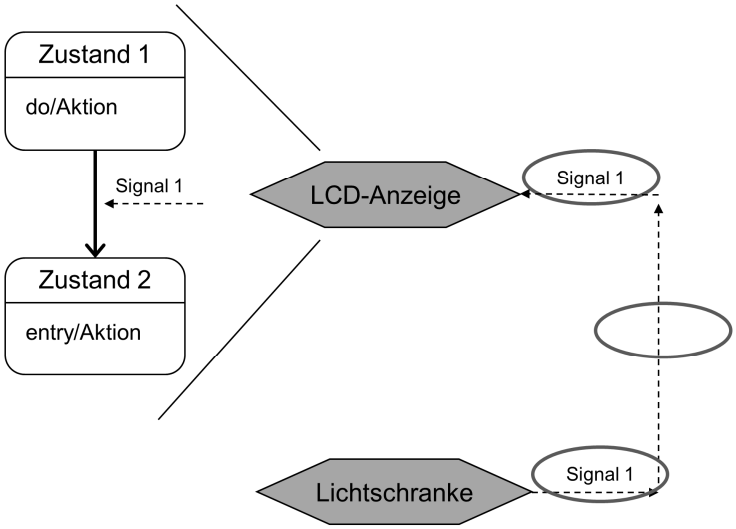


Abbildung 8.14: Lichtschranke und LCD-Anzeige

Im zweiten Beispiel muss die Summe der ausgehenden Leistungen genauso groß sein wie die Summe der eingehenden Leistungen. Diese Bedingung kann mit den bereits vorhandenen Bedingungstypen und Funktionen beschrieben werden. In vielen Fällen wird jedoch nicht die Leistung direkt als Größe angegeben, sondern muss mit Hilfe anderer Größen berechnet werden. Für die elektrische Leistung kann dies beispielsweise durch Multiplikation der Spannung mit der Stromstärke erfolgen. Entsprechende Formeln sind in der globalen Ontologie hinterlegt. Es muss also die Funktion vorhanden sein, die eine gewünschte Größe mit Hilfe der im Modell vorhandenen Größen berechnen kann. Fehlen entsprechende Angaben, so kann die Bedingung und damit die Regel nicht ausgewertet werden. Um sicherzustellen, dass eine Leistungsquelle nicht überlastet wird, wird deren maximale abgebbare Leistung mit der Summe der Leistungen der angeschlossenen Verbraucher verglichen. Dazu sind keine weiteren Bedingungstypen oder Funktionen erforderlich. Abbildung 8.15 zeigt einen Generator (Quelle) mit zwei angeschlossenen elektrischen Motoren (Senken).

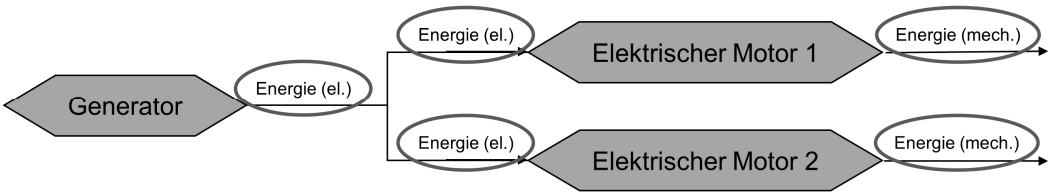


Abbildung 8.15: Generator mit zwei angeschlossenen elektrischen Motoren

Das dritte Beispiel erfordert für die Überprüfung der Abmessungen wiederum einen Zahlenvergleich. Die Ist-Abmessung wird wie im vorherigen Beispiel an Hand der vorhandenen Größen (Einzelabmessungen) berechnet. Der Materialdurchsatz kann ebenfalls mit einem Zahlenvergleich und der Berechnung des Ist-Durchsatzes überprüft werden. Erforderliche Maßnahmen gegen bestimmte Umgebungseinflüsse (z. B. Wasser) können der globalen Ontologie entnommen werden. Ein Vergleich mit den Eigenschaften der eingesetzten Komponenten (z. B. Schutzklasse = IP24) oder des umgebenden Gehäuses zeigt die Einhaltung dieser Anforderung.

Im vierten Beispiel wird die Existenz benötigter Systemelemente mit Hilfe der Anzahl der vorhandenen Systemelemente des bestimmten Typs überprüft werden. In Abbildung 8.16 sind die erforderlichen Systemelemente (Stromquelle und Motorregelung) vorhanden.

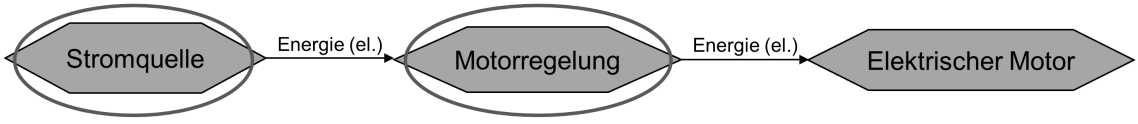


Abbildung 8.16: Elektrischer Motor mit Motorregelung und Stromquelle

Die Stromquelle ist bei einem elektrischen Motor grundsätzlich erforderlich und deren Notwendigkeit wird direkt aus der globalen Begriffswelt abgeleitet. Die Motorregelung dagegen ist nur dann erforderlich, wenn der Motor mit variabler Drehzahl betrieben werden soll. Da in der glo-

balen Begriffswelt bedingte Verknüpfungen nicht ohne weiteres realisierbar sind, wird in der Regel ein veränderter Ausschnitt der Begriffswelt hinterlegt, der die entsprechende Verknüpfung („benötigt“-Verknüpfung zwischen elektrischem Motor und Motorregelung) enthält. Es muss also möglich sein, den Vergleich von Strukturen in einer Bedingung abzubilden. Dies erfolgt nach demselben Prinzip wie die gesamte globale Begriffswelt mit den Modellen verglichen wird, hier jedoch mit einem veränderten Ausschnitt dieser Begriffswelt. Abbildung 8.17 zeigt den für das Beispiel relevanten Teil der globalen Begriffswelt sowie den veränderten Ausschnitt, der zusätzlich für elektrische Motoren mit variabler Drehzahl gilt.

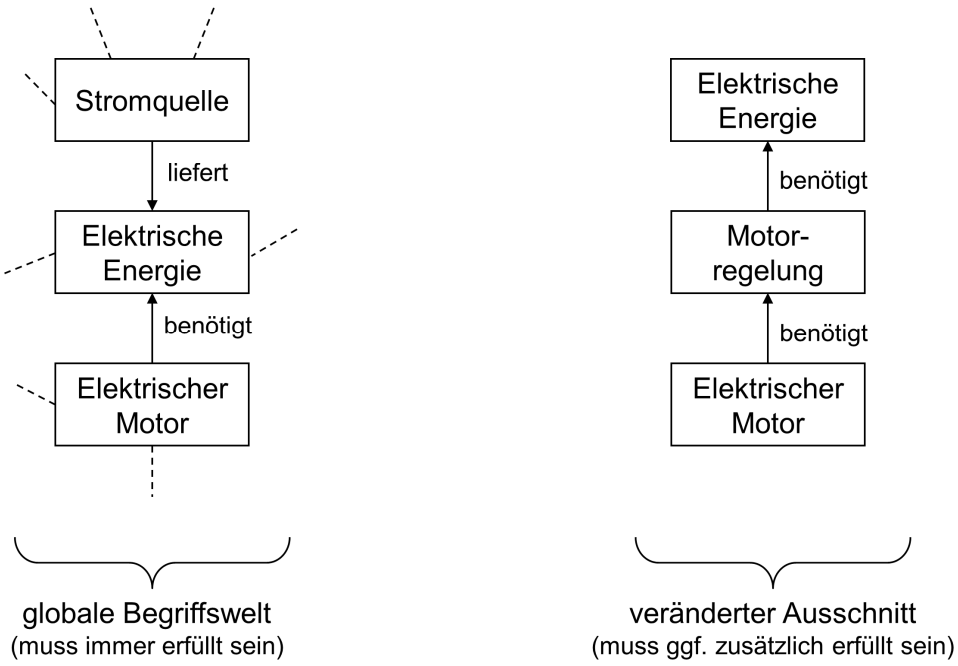


Abbildung 8.17: Globale Begriffswelt und veränderter Ausschnitt für elektrische Motoren variabler Drehzahl

Mit diesen Erkenntnissen wird der Aufbau der Regeln erweitert und sieht nun wie in Abbildung 8.18 dargestellt aus. Die Regel erhält das zusätzliche Attribut „Inaktiv“, um die Regel bei Bedarf deaktivieren bzw. ignorieren zu können.

Die Bedingungstypen *Stringvergleich* und *Strukturvergleich* ergänzen die für die strukturelle Prüfung erforderlichen Bedingungstypen, damit die Anforderungen der inhaltlichen Prüfung erfüllt werden können. Der *Stringvergleich* funktioniert ähnlich dem *Zahlenvergleich*, jedoch können zwei Strings lediglich gleich oder ungleich sein. Der *Strukturvergleich* bezeichnet den Vergleich des Modells mit einem veränderten Ausschnitt der globalen Begriffswelt. Das Attribut *Struktur* verweist auf die Ontologie, in der der veränderte Ausschnitt modelliert ist.

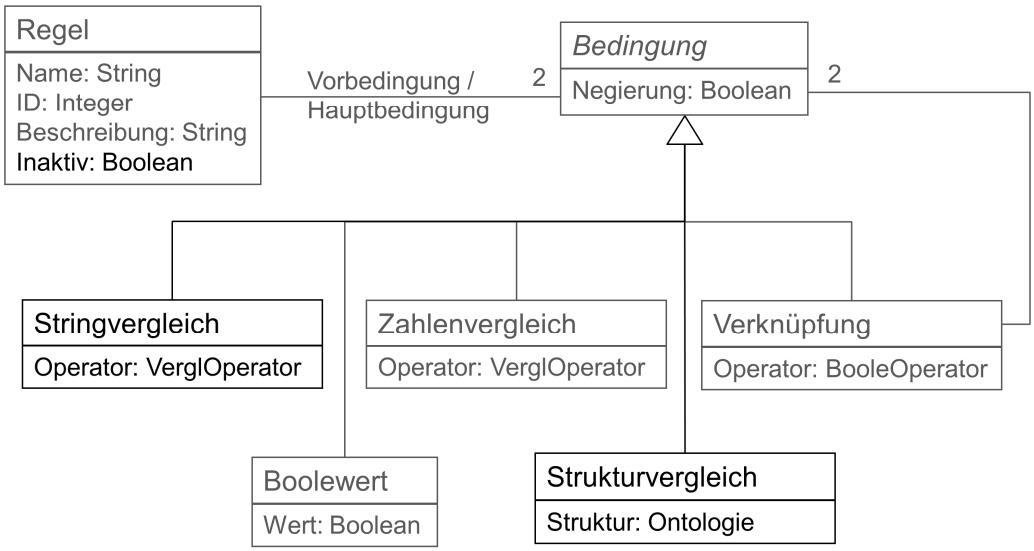


Abbildung 8.18: Erweiterter Aufbau von Regel mit Bedingungstypen

Der Bedingungstyp *Zahlenvergleich* wird, wie in Abbildung 8.19 dargestellt, ebenfalls erweitert. Zur bisher vorhandenen Beschreibung einer Anzahl von Elementen werden die Summation einer beliebigen Anzahl von Werten (*Summe*) und die Ermittlung des Minimums bzw. des Maximums von einer Gruppe von Werten hinzugefügt. Wie bereits bei der Anzahl gibt die *Quelle* an, welche Gruppe von Werten einfließt. Ebenfalls eine *Quelle* besitzt die *Größe*, wobei die Quelle in diesem Fall lediglich auf einen einzigen Wert verweisen darf. Die Größe beschreibt beispielsweise eine physikalische Größe, die in einer Eigenschaft eines Modellelements abgelegt sein kann (z. B. Spannung einer elektrischen Energiequelle). Ist die Größe nicht im Modellelement abgelegt, kann sie gegebenenfalls mit Hilfe von Formeln aus anderen Größen, die im Modellelement vorhanden sind, berechnet werden.

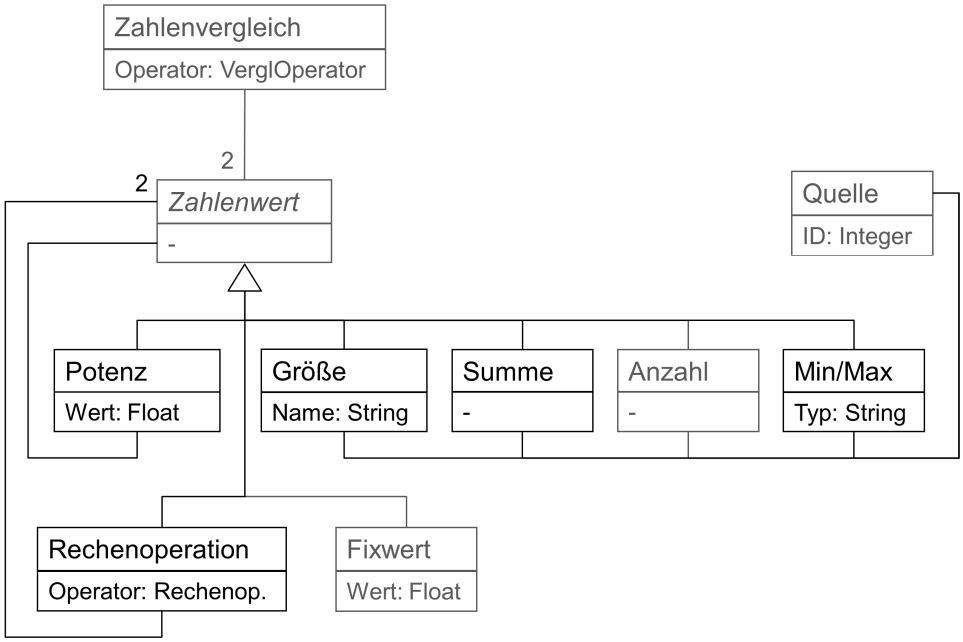


Abbildung 8.19: Erweiterter Ausschnitt des Bedingungstyps Zahlenvergleich

Um einfache Rechenoperationen zu beschreiben, wurden die *Rechenoperation* und die *Potenz* ergänzt. Mit Hilfe der Rechenoperationen lassen sich zwei Zahlenwerte gemäß den mathematischen Grundrechenarten verknüpfen, die Potenz ermöglicht die Potenzierung eines Zahlenwerts. Diese beiden Elemente dienen lediglich dazu, Zahlenwerte miteinander zu verknüpfen. Komplexere Formeln, die zwei oder mehrere Größen zu einer neuen Größe verknüpfen, werden in der globalen Begriffswelt abgebildet und es wird nur die (Ergebnis-) Größe im Zahlenvergleich eingesetzt (vgl. Größe).

Gemäß dem vorgestellten Aufbau lassen sich nahezu beliebige Regeln erstellen. Dabei wird ein Teil der Regeln, wie beispielsweise die Notwendigkeit weiterer Systemelemente (*benötigt*-Verknüpfung), automatisiert aus der globalen Ontologie abgeleitet und müssen nicht manuell gepflegt werden. Einige der Regeln (z. B. Energieerhaltungssatz) können aus den vorhandenen Informationen nicht automatisiert abgeleitet und müssen explizit definiert werden. Bei Bedarf lassen sich die Bedingungstypen erweitern, indem neue Typen wie in den Beispielen beschrieben ergänzt werden. Wichtig dabei ist, dass die zugehörige Auswertefunktionalität im Regelinterpreter hinterlegt ist, damit die Auswertung erfolgen kann.

Damit die Regeln sinnvoll und effektiv angewendet sowie einfacher gepflegt werden können, ist es erforderlich diese zu strukturieren. Im Folgenden wird die Strukturierung und Benutzung der Regeln vorgestellt.

### 8.4.3 Zuordnung der Regeln

Die Regeln betreffen jeweils einen Begriff der globalen Begriffswelt. Dass ein Energieverbraucher eine Energiequelle benötigt, ist beispielsweise dem Begriff *Energieverbraucher* zugeordnet bzw. wird mit Hilfe der Relation *Energieverbraucher – benötigt – Energiequelle* abgeleitet. Regeln, die nicht automatisiert abgeleitet sondern explizit hinterlegt werden müssen, werden ebenfalls dem entsprechenden Begriff zugeordnet. Für eine Regel müssen somit lediglich Ableitungen des zugehörigen globalen Begriffs in den Modellen ermittelt werden und die Regel kann auf die entsprechenden Modellelemente angewendet werden. So wird eine Regel beispielsweise nicht nur auf einen Motor angewendet, sondern auch auf einen elektrischen Motor oder einen Synchronmotor (vgl. Abbildung 8.20). Da ein Begriff mehrere Unterklassen besitzen kann, kann die Regel auch auf mehrere Unterbegriffe angewendet werden. Alle Vererbungslinien des Baumes werden bis zu den Blättern verfolgt und die Regel muss auf sämtliche Unterbegriffe bzw. die zugehörigen Modellelemente angewendet werden.

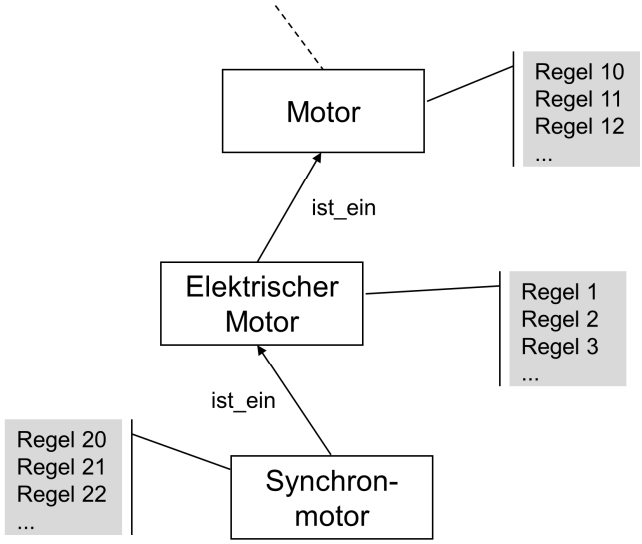


Abbildung 8.20: (Mehrfach-) Anwendung von Regeln auf Grund von Unterbegriffen

Um die Anwendung der Regeln zu vereinfachen und Regeln für bestimmte Modellelemente deaktivieren zu können, werden bei jedem Modellelement die jeweiligen Regelanwendungen hinterlegt. So kann bei einer weiteren Prüfung die Anwendung einer Regel blockiert werden. Abbildung 8.21 zeigt ein entsprechendes Beispiel. Das Modell befindet sich zwar in der allgemeinen Meta-Meta-Ebene, zur besseren Unterscheidung wurde für die Darstellung jedoch die Notation der Partialmodelle verwendet. Die von einem Benutzer (in der Rolle eines Systemverwalters) gesperrte Regel 2 ist grau dargestellt und wird damit bei der Prüfung nicht berücksichtigt.

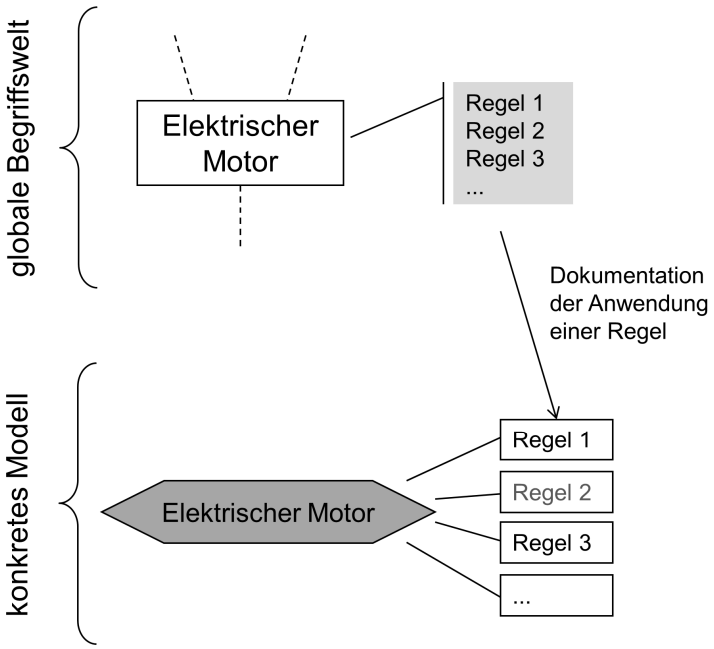


Abbildung 8.21: Hinterlegung von Regelanwendungen in einem konkreten Modell

## 8.5 Aufbau der globalen Ontologie

Die globale Begriffswelt sowie die inhaltlichen Regeln werden in der globalen Ontologie abgebildet. Diese Ontologie ist systemweit verfügbar und enthält alles Wissen, das nicht modellspezifisch ist. Da diese Ontologie sehr umfangreich ist, bietet es sich wiederum an, diese in mehrere, miteinander verknüpfte Teilontologien aufzuteilen. Der Aufwand zur Wartung und Pflege dieser großen Ontologie kann so besser beherrscht werden. Trotzdem ist es möglich eine einzige Ontologie zu erstellen.

Je nach Anwendungsgebiet für die Prüfung muss die Ontologie die Begriffe der gesamten Mechatronik oder nur von Teilbereichen enthalten. Dabei kann es sinnvoll sein, die einzelnen Disziplinen aufzuteilen und beispielsweise physikalische Zusammenhänge ebenfalls getrennt zu betrachten. Es ist jedoch darauf zu achten, dass die einzelnen Teilontologien nicht völlig losgelöst voneinander entwickelt werden und besonderes Augenmerk auf die Verknüpfung der Ontologien gelegt wird.

Jede der Teilontologien kann weiter strukturiert werden, indem die Begriffe, die Elemente, Sachverhalte oder Fakten beschreiben (z. B. *elektrischer Motor – benötigt – elektrische Energiequelle*) von physikalischen Größen und Einheiten getrennt werden. Ebenso können die explizit definierten Regeln in einer separaten Teilontologie abgebildet werden. Die Ontologie lässt sich dann wie in Abbildung 8.22 dargestellt strukturieren.

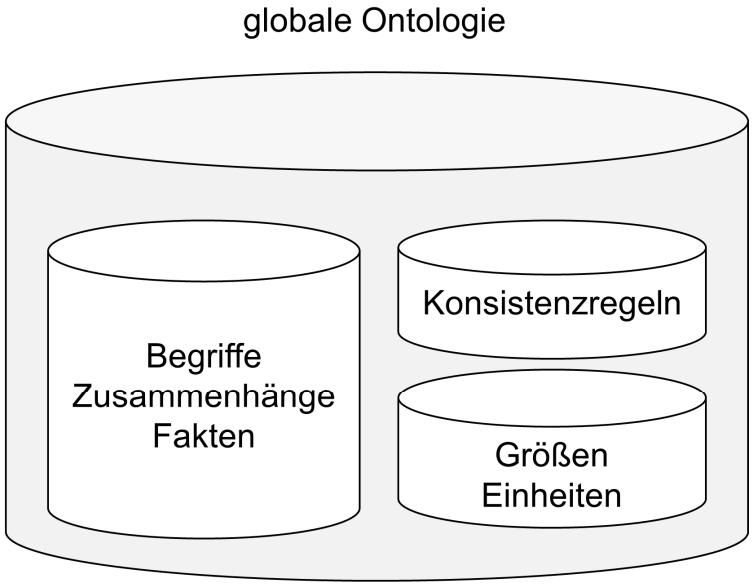


Abbildung 8.22: Mögliche Strukturierung der globalen Ontologie

Die Begriffe, Zusammenhänge und Fakten bilden den Kern der Ontologie und machen den größten Teil der globalen Begriffswelt aus. Die Regeln beziehen sich auf die Begriffe. Die Größen und Einheiten ergänzen die Begriffe. Die hier vorgestellte Aufteilung ist nicht zwingend erforderlich, erlaubt es jedoch statt einer großen, mehrere kleinere Ontologien zu verwalten.

Prinzipiell ist es möglich, neben den Begriffen auch entsprechende Synonyme in die globale Ontologie zu integrieren. Dies reduziert den Aufwand zur Erstellung der lokalen Ontologien, sofern das zugehörige Modell in der globalen Ontologie definierte Synonyme enthält. Hierbei ist jedoch abzuwägen, ob die Verkleinerung der lokalen Ontologien die Vergrößerung der globalen Ontologie rechtfertigt, zumal realistisch gesehen nie alle möglichen Synonyme hinterlegt werden können.

Bei der Erstellung der globalen Ontologie ist darauf zu achten, dass diese alle Begriffe enthält, die notwendig sind, um den gesamten Beschreibungsbereich der zu prüfenden Modelle abdeckt. Eine Abdeckung aller Begriffe ist mit vertretbarem Aufwand nicht möglich. Deshalb ist die Erstellung einer reduzierten globalen Ontologie, die einen gewissen Bereich der Mechatronik, beispielsweise pneumatische Systeme oder die Mikrosystemtechnik, abdeckt, anzustreben.

In diesem Kapitel wurde das Konzept zur Wissensrepräsentation mit Hilfe mehrerer modelltypspezifischer lokaler Ontologien und einer globalen Ontologie vorgestellt. Die Übersicht hat gezeigt, dass eine gemeinsame globale Ontologie notwendig ist, um den Inhalt mehrerer Modelle in Zusammenhang zu bringen. Bei heterogenen Modellen muss für jeden Modelltyp jedoch zusätzlich eine Beschreibung existieren, die es ermöglicht, die Aussagen des jeweiligen Modells auf die globale Begriffsebene zu heben und damit zu interpretieren. Diese Beschreibung besteht aus einer strukturellen und aus einer inhaltlichen Beschreibung und wird in Form der lokalen Ontologie bereitgestellt. Für die Prüfung selbst sind Regeln erforderlich, deren Aufbau vorgestellt wurde. Abgelegt sind diese in der globalen Ontologie, wo sie den dort definierten Begriffen zugeordnet sind. Teilweise sind sie implizit in den Begriffsverknüpfungen vorhanden und können von dort abgeleitet werden, teilweise müssen sie explizit definiert werden. Auf Grund des Umfangs der globalen Ontologie ist zudem eine entsprechende Strukturierung erforderlich, um diese beherrschbar zu halten. Letztendlich ist es möglich, die Aussagen, die in den verschiedenen Modellen enthalten sind, in die globale Begriffswelt zu transformieren und die dort definierten Regeln anzuwenden. Im folgenden Kapitel wird gezeigt, wie dies in einem Prototyp umgesetzt ist.

# 9 Realisierung eines Prototyps

Der Prototyp soll in vereinfachter Form die Funktionalität des vorgestellten Konzepts zur automatisierten Prüfung heterogener Modelle aufzeigen und besteht aus mehreren Komponenten. Ein einfacher Editor für die Modelle dient dazu, diese für den Benutzer graphisch darzustellen. Im produktiven Einsatz wird er durch die gängigen Entwicklungswerkzeuge ersetzt werden. Das Agentensystem, welches die eigentliche Prüfung durchführt, wird unter Einsatz eines Agentenframeworks realisiert. Dafür wird das Verhalten der Agenten in Behaviours abgebildet. Der Koordinierungsagent besitzt eine graphische Benutzungsschnittstelle. Mit ihrer Hilfe kann ein Benutzer die Prüfung anstoßen und bekommt die Ergebnisse präsentiert. Die Ontologien (global und lokal) enthalten das Wissen über die Mechatronik und die Modelle. Sie sind im Prototyp auf den notwendigen Inhalt zur Präsentation der Funktionalität des Konzepts beschränkt und werden mit dem Werkzeug protégé erstellt. Zunächst wird jedoch der Editor für die mechatronischen Modelle vorgestellt.

## 9.1 Editor für die mechatronischen Modelle

Als Engineering Werkzeug wird ein einfacher Editor für die Beispielmodelle benutzt. Er basiert auf der Entwicklungsumgebung Eclipse [Ecli14], dem Eclipse Modeling Framework [EMF14] und dem Graphical Modeling Framework [GMF14]. Mit dem Editor können die Modelle benutzerfreundlich erstellt bzw. verändert werden und er ist in der Lage, die Modelle als XML-Dateien abzuspeichern. Diese können über eine Dateischnittstelle von den Modellagenten eingelesen werden. Auf eine direkte Datenschnittstelle zwischen Engineering Werkzeug und Modellagent wird deshalb verzichtet. Weitergehende Informationen zur Erstellung eines solchen Editors können [Gram11] entnommen werden. Abbildung 9.1 zeigt die Benutzungsoberfläche des Editors.

Im Modellexplorer (links) sind die vom Benutzer erstellten Modelle gelistet und können dort verwaltet werden. Sie sind dort nach Modelltyp gruppiert. In der Mitte befindet sich der Editierbereich, in dem die Modelle erstellt und verändert werden können. Unten links wird eine Übersicht über das gerade geöffnete Modell dargestellt. Im Eigenschaftsfenster (unten) können die Details der eingesetzten Modellelemente bearbeitet werden. Teilweise sind die Eigenschaften frei editierbar, teilweise können die Werte aus einer Liste ausgewählt werden. Die Modellelemente selbst entstammen der Palette (rechts), die diese zur Verfügung stellt.

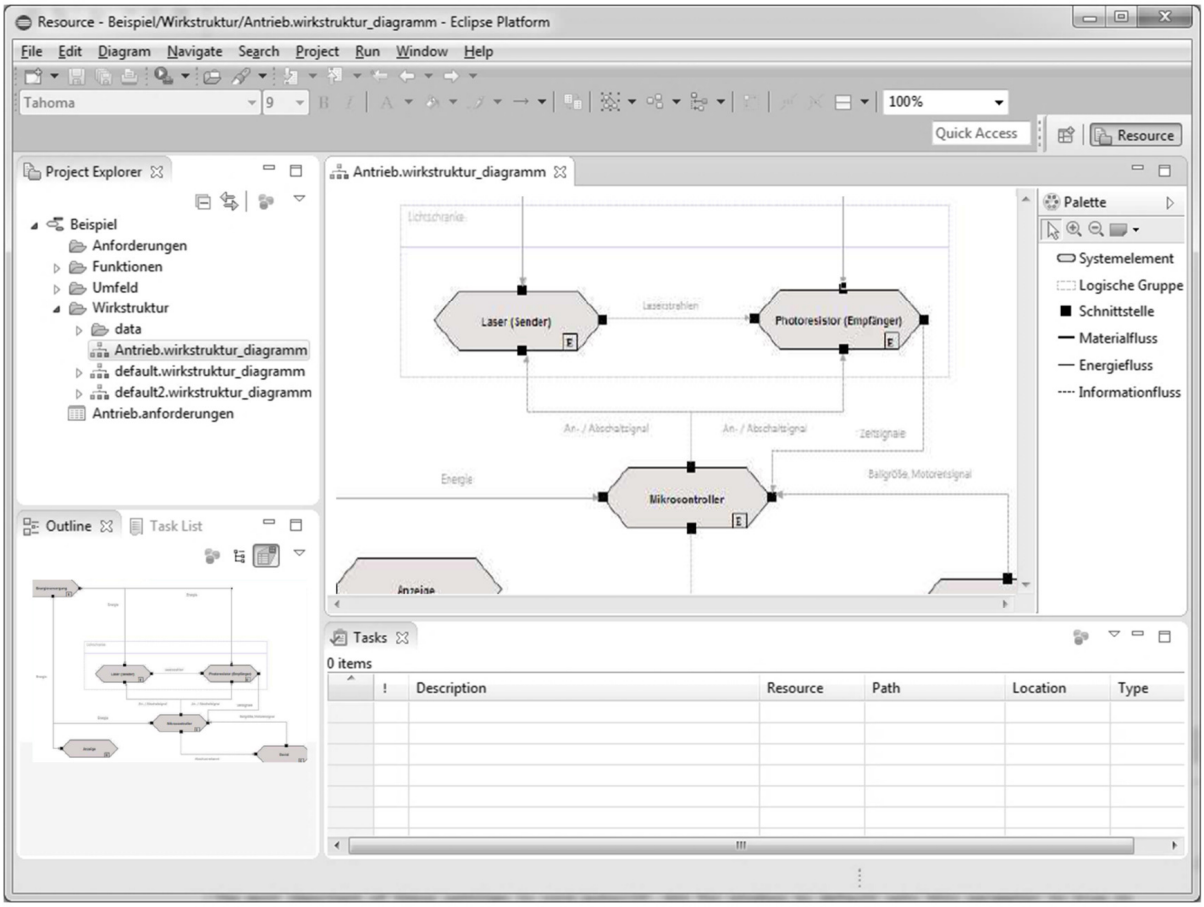


Abbildung 9.1: Benutzungsoberfläche des Editors

Die Heterogenität der Modelle, die für die Evaluation notwendig ist, entsteht durch die Erstellung verschiedener Modelle (Wirkstruktur, Zustandsdiagramm und Funktionsstruktur) und durch die Verwendung unterschiedlicher Begriffswelten.

## 9.2 Implementierung des Agentensystems

Die Agenten des Prototyps sind mit Hilfe der Agentenplattform JADE in der Programmiersprache Java realisiert. Durch den Einsatz vorgefertigter Elemente reduziert sich der Implementierungsaufwand, jedoch existieren gewisse Vorgaben, die bei der Implementierung berücksichtigt werden müssen. So besitzen Agenten in JADE sogenannte Behaviours, die das Verhalten des Agenten definieren und von diesem nacheinander abgearbeitet werden. Behaviours können dabei einmalig oder nach der Bearbeitung erneut eingeordnet und damit wiederholt ausgeführt werden. Jedoch ist es nicht immer sinnvoll, alle Aufgaben eines Agenten in einem separaten Behaviour umzusetzen, andererseits kann es schnell unübersichtlich werden, wenn einzelne Behaviours zu groß und zu komplex sind. Im Folgenden wird vorgestellt, wie die vier Agententypen im Prototyp umgesetzt wurden.

### 9.2.1 Realisierung der Modellagenten

Ein Modellagent besitzt drei Behaviours, die jedoch nicht alle gleichzeitig ausgeführt werden.

- *Initialisierungs-Behaviour*: Die Startphase des Modellagenten wird über dieses Behaviour gesteuert. Es ist das erste Behaviour, das ausgeführt wird und initialisiert das Modell. Zudem wird der Modellagent beim Directory Facilitator der Agentenplattform (DF) registriert. Dieses Behaviour wird nur einmalig bei der Initialisierung des Modellagenten ausgeführt und aktiviert als letzte Aktion das Informationsermittlungs-Behaviour.

- *Informationsermittlungs-Behaviour (Modell)*: Die Rolle des Informationsermittlers ist in diesem Behaviour umgesetzt. Kommt dieses zur Ausführung, so werden sämtliche Informationsanfragen, die sich seit der letzten Ausführung in der Warteschlange befinden, ausgeführt und beantwortet. Dieses Behaviour wird dementsprechend während der gesamten Lebenszeit des Modellagenten (nach der Initialisierungsphase) zyklisch aufgerufen.

- *Schnittstellenwächter-Behaviour*: Das Behaviour des Schnittstellenwächters kommt nur dann zum Einsatz, wenn eine permanente Prüfung gewünscht ist. Es wird dann zyklisch aufgerufen und prüft das Modell im Engineering Werkzeug auf Änderungen. Werden Änderungen erkannt, wird der Koordinierungsagent informiert, der daraufhin eine Prüfung einleitet.

Die Funktionalität des Ontologieinterpreters, der die Schnittstelle zur lokalen Ontologie bildet, ist in Form eines Objekts, das der Agent besitzt und auf das die Behaviours Zugriff haben, umgesetzt. Der Anfrageinterpreter und der Modelltransformator sind ebenfalls als Objekte realisiert, um den Umfang der Behaviours in Grenzen zu halten. Die bereitgestellten Funktionen können so von den Behaviours genutzt werden, sind selbst jedoch nicht als eigenständige Behaviours realisiert, da sie kein eigenständiges Verhalten benötigen.

Das Abbild des Modells wird vom Initialisierungs-Behaviour unter Nutzung der Funktionen des Modelltransformators und des Ontologieinterpreters als Objektstruktur erstellt. Das Modell selbst liegt im Prototyp als XML-Datei vor, die vom Modelltransformator eingelesen und in die Meta-Meta-Modell-Struktur umgewandelt wird. Diese Struktur ist dem Modellagenten zugeordnet und die verschiedenen Behaviours können darauf zugreifen.

### 9.2.2 Realisierung der Regelagenten

Die Regelagenten erhalten ebenfalls drei Behaviours.

- *Verwaltungs-Behaviour*: Dieses Behaviour verwaltet den Regelagenten und empfängt beispielsweise die Nachrichten des Koordinierungsagenten. Es stößt die Prüfung nach einem eingegangenen Auftrag an und ist in der Lage, Rückmeldung über den aktuellen Zustand des Regelagenten zu geben. Dieses Behaviour wird zyklisch aufgerufen.

- *Initialisierungs-Behaviour*: Die Startphase des Regelagenten wird über dieses Behaviour gesteuert. Es ist das erste Behaviour, das ausgeführt wird und meldet den Regelagenten beim DF an. Dieses Behaviour wird nur einmalig bei der Initialisierung des Regelagenten ausgeführt und aktiviert als letzte Aktion das Verwaltungs-Behaviour.

- *Prüfungs-Behaviour*: Dieses Behaviour führt die eigentliche Prüfung (Rolle des Regelinterpreters) aus und wird bei einer entsprechenden Anfrage durch das Verwaltungs-Behaviour aktiviert. Es ermittelt an Hand der zu prüfenden Modelle auf welche Modelle bzw. Modellelemente die Regel angewendet werden muss, prüft deren Einhaltung und sendet das Ergebnis nach abgeschlossener Prüfung an den Koordinierungsagenten. Bei jeder Ausführung des Behaviours wird die Regel nur einmal angewendet. So ist es möglich, nach jeder Regelanwendung das Verwaltungs-Behaviour auszuführen und der Agent bleibt reaktionsfähig. Ansonsten wäre der Agent bis zur vollständigen Prüfung aller Modelle bezüglich seiner Regel blockiert. Ein Ausschnitt dieses Behaviours ist in Abbildung 9.2 dargestellt.

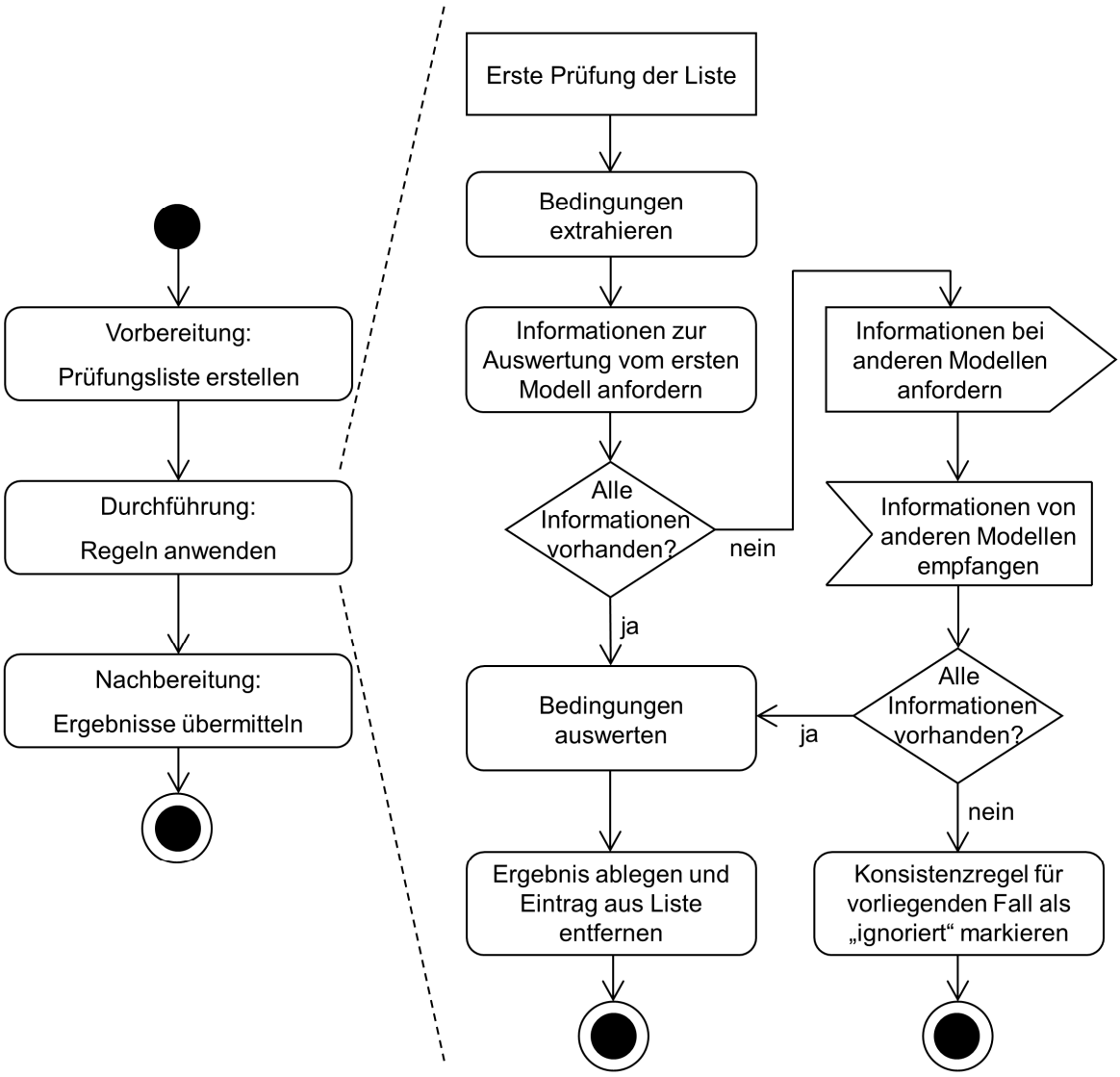


Abbildung 9.2: Ausschnitt des Prüfungs-Behaviours

### 9.2.3 Realisierung des Ontologieagenten

Der Regelagent stellt die Schnittstelle zur globalen Ontologie dar und besitzt drei Behaviours.

- *Initialisierungs-Behaviour*: Die Startphase des Ontologieagenten wird über dieses Behaviour gesteuert. Es ist das erste Behaviour, das ausgeführt wird. Die Ontologie wird eingelesen und der Agent beim DF registriert. Dieses Behaviour wird nur einmalig bei der Initialisierung des Ontologieagenten ausgeführt. Es führt nach der Registrierung einmalig das Regelinstanziierungs-Behaviour aus und aktiviert anschließend das Informationsermittlungs-Behaviour.

- *Regelinstanziierungs-Behaviour*: Dieses Behaviour muss außer beim Systemstart nur dann ausgeführt werden, wenn die globale Ontologie verändert wurde und neue bzw. zusätzliche Regeln abgeleitet werden können. Zuerst wird die Ontologie analysiert und aus den Verknüpfungen der Begriffe werden Regeln abgeleitet. Ebenso werden die explizit hinterlegten Regeln ausgelesen. Die Regeln werden zu Regelobjekten und für jede Regel ein zugehöriger Regelagent instanziiert. Den Agenten wird jeweils eine Regel übergeben.

- *Informationsermittler-Behaviour (Ontologie)*: In diesem Behaviour werden die Anfragen der Regelagenten an die globale Ontologie beantwortet. Dieses Behaviour wird während der gesamten Lebenszeit des Regelagenten (nach der Initialisierungsphase) zyklisch aufgerufen.

Die Funktionalität des Ontologieinterpreters, der die Schnittstelle zur globalen Ontologie bildet, ist in Form eines Objekts, das der Agent besitzt und auf das die Behaviours Zugriff haben, umgesetzt. Die bereitgestellten Funktionen können so von den Behaviours genutzt werden, sind selbst jedoch nicht als eigenständige Behaviours realisiert, da sie kein eigenständiges Verhalten benötigen.

### 9.2.4 Realisierung des Koordinierungsagenten

Der Koordinierungsagent stellt die Schnittstellen zum Benutzer und den Koordinator der Prüfung dar. Dazu besitzt er die folgenden drei Behaviours.

- *GUI-Wächter-Behaviour*: Die Aktionen des Benutzers auf der Benutzungsoberfläche werden an dieses Behaviour gesendet und von diesem bearbeitet. Dazu prüft das Behaviour zyklisch auf neue Aktionen und passt das Systemverhalten entsprechend an. Ebenso werden Informationen aus dem System (z. B. Ergebnisse) auf der Oberfläche präsentiert.

- *Koordinierungs-Behaviour*: Dieses Behaviour wird vom GUI-Wächter-Behaviour aktiviert, wenn der Benutzer eine Prüfung durchgeführt haben möchte. Es ermittelt zunächst vom DF die im System vorhandenen Modell- und Regelagenten und beauftragt letztere mit der Prüfung der Modelle. Anschließend werden die Ergebnisse empfangen und es wird geprüft, ob alle Ergebnisse vorliegen. Diese werden aufbereitet und auf der Benutzungsoberfläche dar-

gestellt. Hier ist wichtig, dass das Behaviour beim Warten auf die Ergebnisse von den Regelagenten den Koordinierungsagenten nicht blockiert und das GUI-Wächter-Behaviour zwischendurch ausgeführt werden kann. Ansonsten erscheint das System aus Sicht des Benutzers zu „hängen“, da die Aktionen auf der Benutzungsoberfläche nicht mehr bearbeitet werden.

- *Änderungswächter-Behaviour*: Wenn von einem Modellagenten Änderungen im Modell festgestellt werden und er diese meldet, so werden sie vom Änderungswächter-Behaviour empfangen und bearbeitet. Dieses Behaviour wird nur benötigt, wenn eine permanente Prüfung gewünscht ist. Im Falle einer Änderung wird das Koordinierungs-Behaviour aktiviert und so eine Prüfung angestoßen. Das Behaviour selbst wird zyklisch ausgeführt, um auf neue Änderungsnachrichten zu prüfen.

Die Aufgabe der Systemverwaltung wird mit Hilfe des in der Agentenplattform vorhandenen DF realisiert. Neuhinzugefügte Modellagenten melden sich und ihre jeweiligen Modelle bei diesem an. Dasselbe gilt für die Regelagenten. Vom DF können die entsprechenden Informationen auch wieder abgefragt werden.

Die Funktionen, die zur Aufbereitung der Ergebnisse für den Benutzer notwendig sind, sind in einem dem Agenten zugeordneten Objekt realisiert, um die Behaviour übersichtlich zu halten.

Die Benutzungsoberfläche des Prüfungssystems ist in Abbildung 9.3 dargestellt. Oben links befinden sich die Steuerelemente, mit denen eine Prüfung gestartet bzw. abgebrochen werden kann. Zusätzlich werden der aktuelle Systemzustand und der Fortschritt der laufenden Prüfung angezeigt. Rechts daneben befindet sich eine Übersicht der im System vorhandenen Modelle. Im unteren Teil werden die Ergebnisse angezeigt.

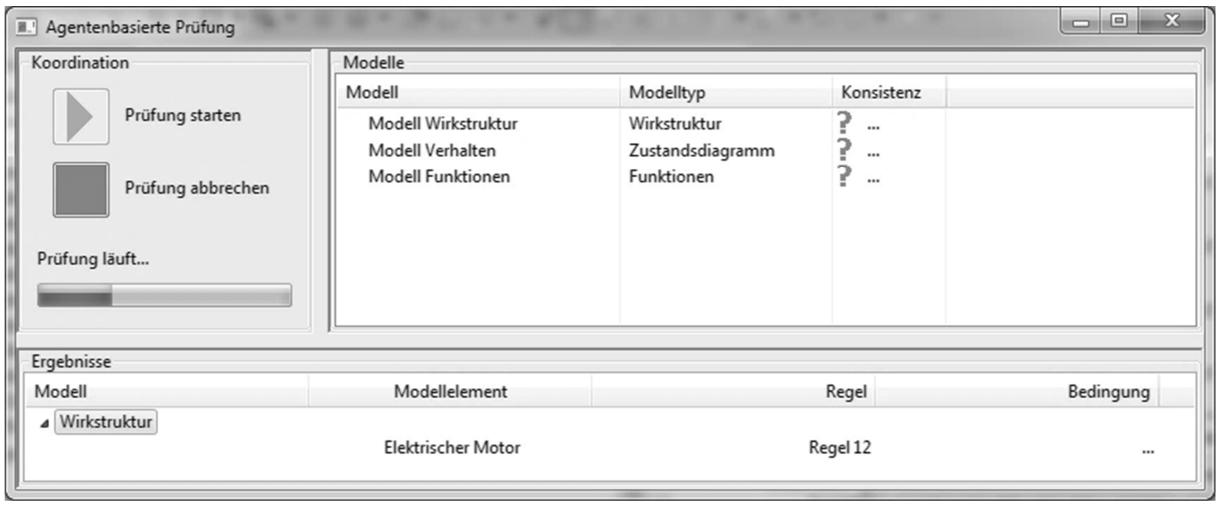


Abbildung 9.3: Benutzungsoberfläche des Prüfungssystems

## 9.3 Realisierung der Ontologien

Die Wissensbasen in Form der globalen und der lokalen Ontologien sind in OWL realisiert. Zur Erstellung wurde das Ontologie-Entwicklungswerkzeug protégé [Stan14] eingesetzt. Dieses unterstützt mehrere Ontologie-Beschreibungssprachen und bietet viele nützliche Funktionen.

Sowohl die globale als auch die lokalen Ontologien wurden so schlank wie möglich gehalten. Es gilt die Open World Assumption (OWA) [Keet13], das heißt, dass nur Aussagen, die in der Ontologie explizit getroffen oder von dieser abgeleitet werden, als wahr oder falsch entschieden werden können. Andere Aussagen, die nicht in der Ontologie enthalten sind, können nicht entschieden werden. Eine Ontologie muss bei dieser Annahme nicht vollständig sein, kann aber einfach erweitert werden.

Die lokalen Ontologien müssen jedoch, bezogen auf die globale Ontologie, vollständig sein, um eine bestmögliche Prüfung zu ermöglichen. Das bedeutet, dass alle lokalen Elemente, die mit Elementen der globalen Ontologie beschrieben werden können, auch beschrieben werden müssen. Ansonsten können Informationen, die im Modell zwar vorhanden sind und auf Grund der Mächtigkeit der globalen Ontologie auch genutzt werden könnten, nicht genutzt werden. Trotzdem gilt auch für die lokalen Ontologien die OWA und Informationen, die nicht definiert sind, werden nicht interpretiert.

Die Regeln entsprechen ebenfalls der OWA, das heißt Regeln, bei welchen nicht alle zur Interpretation notwendigen Informationen aus den Modellen vorliegen, werden nicht ausgewertet.

Bei der Realisierung der Ontologien für den Prototyp wurde die globale Ontologie auf mehrere OWL-Dateien verteilt. So existiert jeweils eine separate Datei für die Definition der expliziten Regeln, der Größen, der mechatronischen Elemente etc. Die so entstandenen Teilontologien werden per Import-Statement in die eigentliche globale Ontologie importiert. Ebenso wird die globale Ontologie in die lokalen Ontologien importiert, um die entsprechenden Definitionen benutzen zu können. Abbildung 9.4 zeigt einen Ausschnitt der globalen Ontologie im Ontologie-Entwicklungswerkzeug protégé.

Einige Begriffe, beispielsweise die in der Mechatronik vorkommenden Größen (physikalische Größen etc.), sind sowohl als Klassen als auch als Instanzen und somit doppelt in der Ontologie hinterlegt. Nur auf diese Weise war es möglich, von den Bedingungen der Regeln, die als Instanzen realisiert sind, auf die jeweiligen Größen zu verweisen.

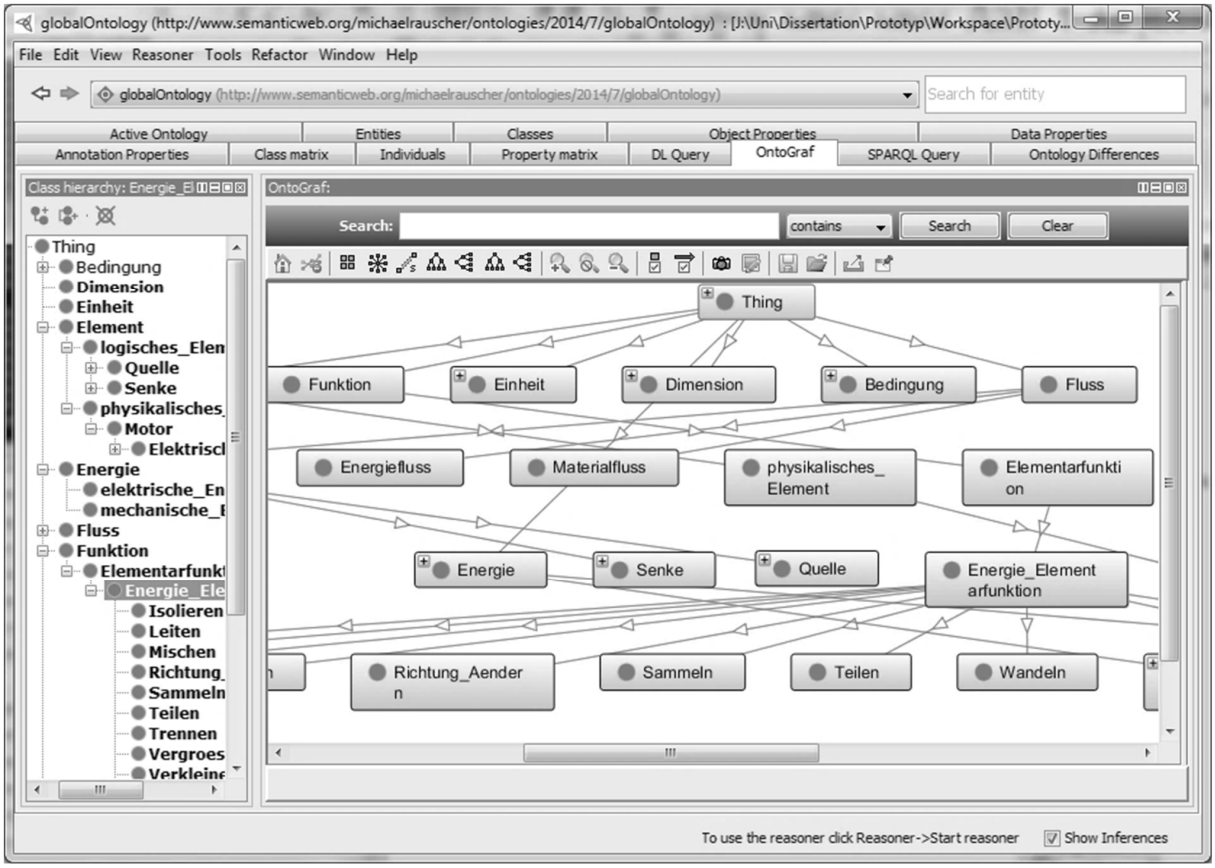


Abbildung 9.4: Ausschnitt der globalen Ontologie im Werkzeug protégé

In diesem Kapitel wurde die Realisierung eines Prototyps für die agentenbasierte Prüfung heterogener Modelle vorgestellt. Zunächst wurde der graphische Editor für die mechatronischen Modelle betrachtet. Er ist mit Hilfe von Eclipse realisiert und erzeugt XML-Dateien für die Modelle. Die Implementierung der Agenten unter Einsatz des Frameworks JADE erfordert die Realisierung des Verhaltens eines Agenten in sogenannten Behaviours. Die für die Umsetzung benötigten Behaviours wurden vorgestellt. Schließlich wurden die Ontologien mit Hilfe des Werkzeugs protégé erstellt. Für diese gilt die OWA. Im folgenden Kapitel wird der entstandene Prototyp in einem Beispielszenario eingesetzt.

# 10 Anwendungsbeispiel für die Prüfung von Modellen

Ein Anwendungsbeispiel, um die praktische Anwendbarkeit des Konzepts und des entstandenen Prototyps aufzuzeigen, erfordert mehrere heterogene Modelle. Dazu wird ein Szenario erstellt, welches aus drei Modellen besteht. Die Durchführung der Prüfung wird beobachtet und die Ergebnisse werden erfasst. Schließlich werden die Ergebnisse bewertet und die Erfahrungen geschildert.

## 10.1 Beispielszenario

Das Szenario, an welchem das Konzept beispielhaft angewendet wird, besteht aus drei Modellen. Alle drei Modelle, Wirkstruktur, Funktionsstruktur und Zustandsdiagramm (als Bestandteil des Verhaltens), beschreiben jeweils einen Ausschnitt desselben mechatronischen Systems. Jedes Modell besitzt eine lokale Ontologie, welche den vorliegenden Modelltyp beschreibt, und wird von einem Modellagenten repräsentiert. Diese sind auf Grund der unterschiedlichen Schnittstellen zu den Modellen und der notwendigen Modelltransformation in die Meta-Meta-Ebene modellspezifisch, ansonsten jedoch vom allgemeinen Modellagenten abgeleitet. Abbildung 10.1 zeigt einen Überblick über das Szenario.

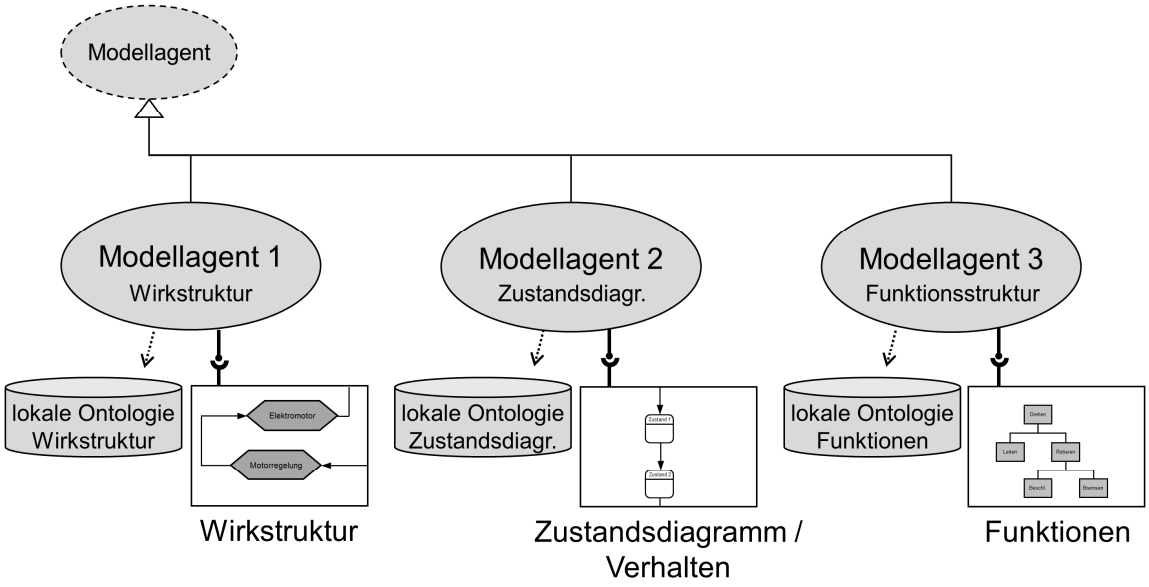


Abbildung 10.1: Überblick über das Beispielszenario

Das Szenario beschreibt einen Ausschnitt aus dem Antrieb des automatisierten Fußballschuh David [RJG11]. Der Ausschnitt der Wirkstruktur (siehe Abbildung 10.2) enthält die Systemelemente *Elektromotor*, *Motorregelung*, *V-Regelung* (Geschwindigkeitsregelung) und *Notaus-Taster*. Die Flüsse und die entsprechenden Schnittstellen inklusive verschiedener hinterlegter Eigenschaften sind ebenfalls enthalten. Es wurden, unter anderem mit dem Begriff *Elektromo-*

*tor*, Begriffe gewählt, die nicht den Begriffen der globalen Ontologie entsprechen, um die Heterogenität der Modelle zu unterstreichen.

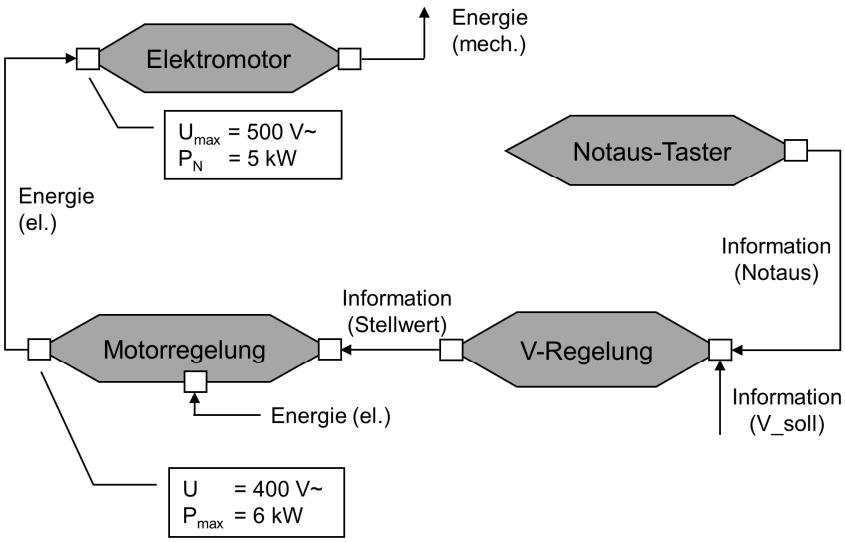


Abbildung 10.2: Beispielszenario: Wirkstruktur (Ausschnitt)

Das Zustandsdiagramm (siehe Abbildung 10.3) beschreibt das Verhalten der Geschwindigkeitsregelung und besteht aus den beiden Zuständen *Normalbetrieb* und *Nothalt*. Der Trigger für den Zustandsübergang ist das Notaus-Signal. Die Zustände selbst sind für das vorliegende Szenario nicht relevant.

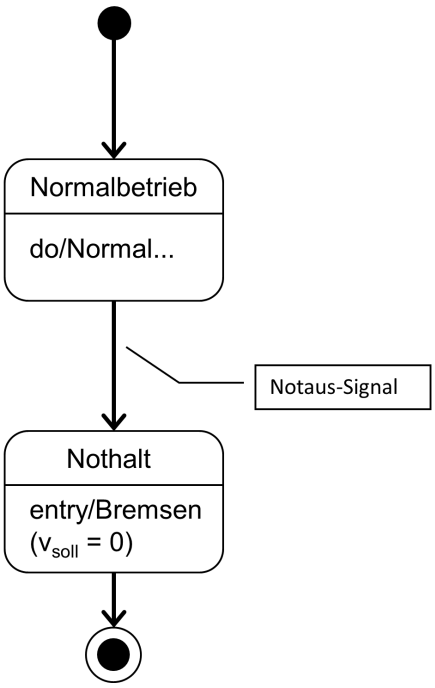


Abbildung 10.3: Beispielszenario: Zustandsdiagramm

Die Funktionsstruktur (siehe Abbildung 10.4) enthält einen Ausschnitt mit den Funktionen *Band bewegen*, *El. Energie wandeln* und *Mech. Energie wandeln*.

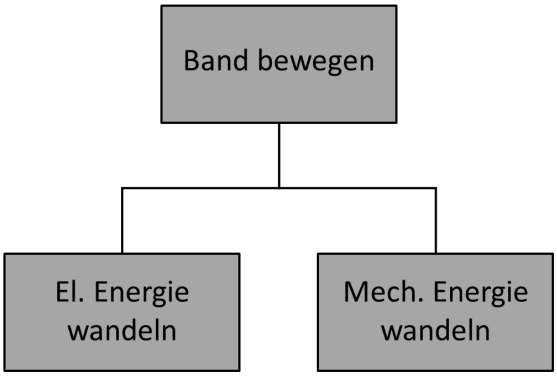


Abbildung 10.4: Beispielszenario: Funktionsstruktur

Alle drei Modelle liegen in Form einer XML-Datei vor und können so vom jeweiligen Modellagenten eingelesen werden.

Neben den szenariospezifischen Modellagenten existieren der Koordinierungsagent, der Ontologieagent und der Directory Facilitator (DF) (vgl. Abbildung 10.5).

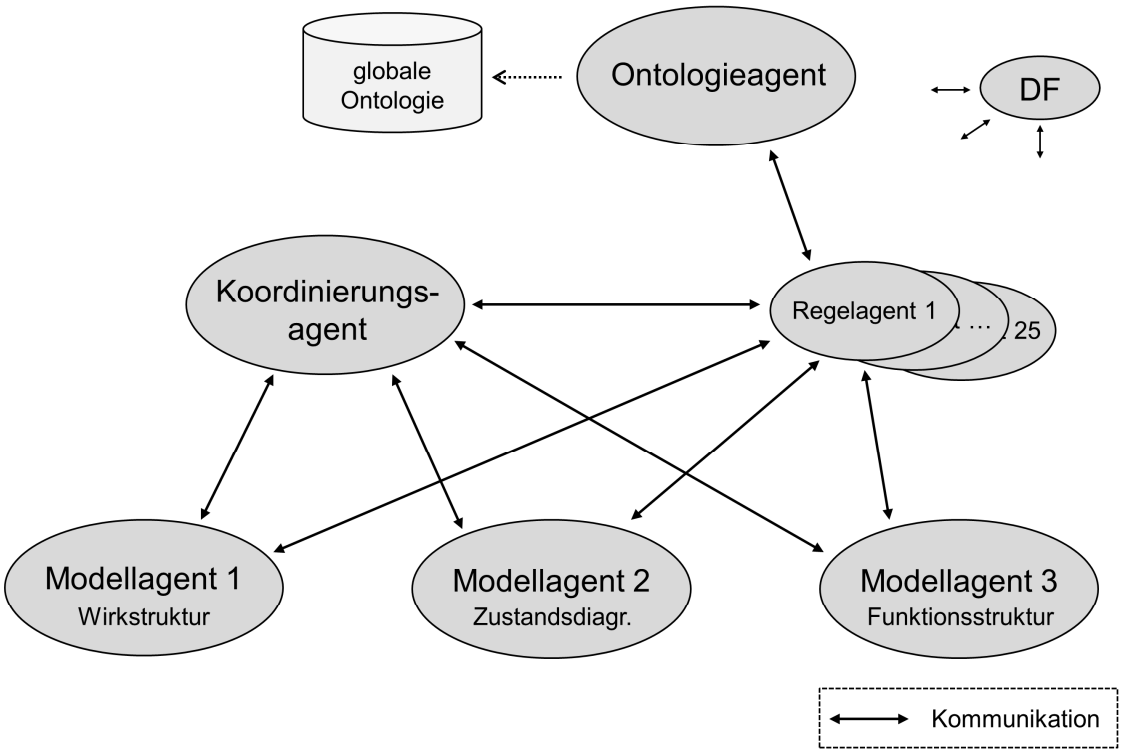


Abbildung 10.5: Agenten im Beispielszenario

Die globale Ontologie enthält etwa 300 Begriffe, aus denen 25 Regeln abgeleitet wurden. Dementsprechend existieren 25 Regelagenten und das System enthält insgesamt 31 Agenten.

## 10.2 Ablauf der Prüfung

Nachdem das Prüfungssystem gestartet wurde und alle Agenten initialisiert sind, wird vom Benutzer eine Prüfung angefordert. Der Koordinierungsagent empfängt den entsprechenden Auftrag von der GUI und ermittelt die im System vorhandenen Modelle bzw. deren Modellagenten. Anschließend ermittelt er die Regeln und Regelagenten. Diese beauftragt er mit der Durchführung einer Prüfung der Modelle und wartet anschließend auf die Ergebnisse. Währenddessen wird der Benutzer über die GUI über den aktuellen Stand der Prüfung informiert. Die Regelagenten führen die Prüfung durch, indem sie die Regeln unter Zuhilfenahme der Modellagenten anwenden. Dazu senden sie zunächst Anfragen an alle Modellagenten, in denen sie nach Modellelementen, auf die die jeweilige Regel anzuwenden ist, suchen. Wenn sie eine entsprechende Rückmeldung bekommen und solche Modellelemente im Modell existieren, versuchen sie, alle weiteren Informationen die Regel betreffend zu ermitteln. Dies geschieht ebenfalls über entsprechende Anfragen an die Modellagenten. Können alle Informationen ermittelt werden, wird die Regel ausgewertet. Fehlen Informationen, so kann die Regel für das zugehörige Modellelement nicht ausgewertet werden, was jedoch ebenfalls als Ergebnis festgehalten wird. Die Ergebnisse der Prüfung werden schließlich an den Koordinierungsagenten übermittelt, welche dieser dem Benutzer bereitstellt. Der Benutzer bekommt diese, wie in Abbildung 10.6 dargestellt, präsentiert.

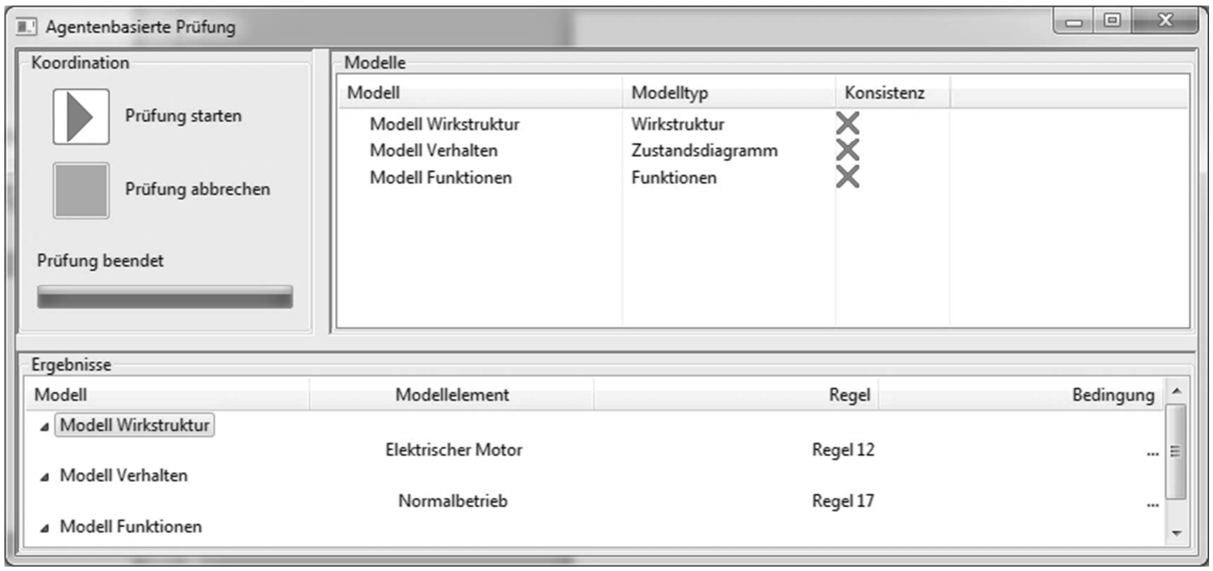


Abbildung 10.6: Darstellung der Prüfungsergebnisse für den Benutzer

## 10.3 Ergebnisse und Erfahrungen

Bei der Prüfung des Beispielszenarios kamen 25 Regeln zum Einsatz. Beispiele dafür sind die Überprüfung der Quelle und der Senke des Notaus-Signals (modellübergreifend, siehe Abbil-

dung 10.7) oder der Ausgangsspannung der Motorregelung und die Eingangsspannung des Elektromotors (modellintern, siehe Abbildung 10.8).

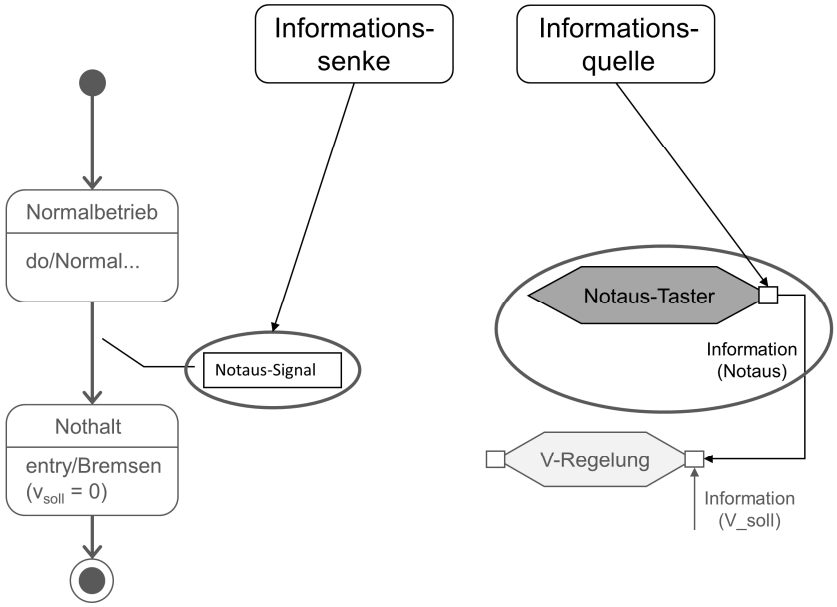


Abbildung 10.7: Beispiel für eine modellübergreifend angewendete Regel

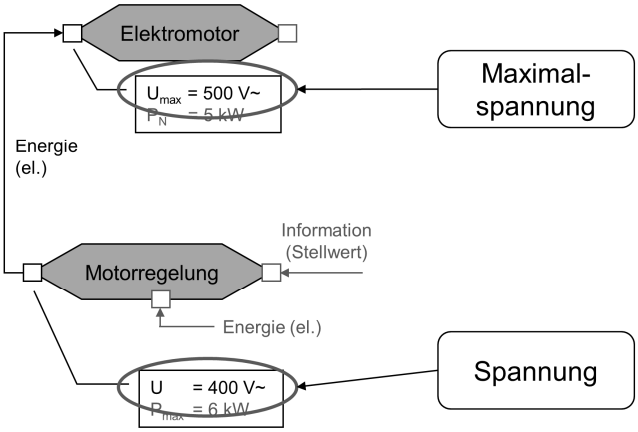


Abbildung 10.8: Beispiel für eine modellintern angewendete Regel

20 der überprüften Regeln wurden für die modellinterne Anwendung vorgesehen, 5 davon betrafen modellübergreifende Sachverhalte. Zwei Regeln konnten nicht überprüft werden, da sie auf Grund fehlender Informationen in den Modellen nicht angewendet werden konnten. Beispielsweise konnte nicht überprüft werden, ob der Elektromotor in der Wirkstruktur ein ausreichendes Drehmoment liefert, da das angetriebene Element nicht im Modell vorhanden ist. Zwei Regeln wurden nicht erfüllt und als Fehler bzw. Inkonsistenzen gemeldet. So wird zum Beispiel die Funktion *Translation erzeugen* von keinem Systemelement der Wirkstruktur bereitgestellt.

Die Prüfung benötigte auf dem zur Verfügung stehenden Standard-PC mit 2,6 GHz für das Beispielszenario nur wenige Sekunden. Es ist jedoch zu erwarten, dass diese bei komplexen Szenarien mit mehr und umfangreicheren Modellen deutlich mehr Zeit in Anspruch nimmt. Da jeder

Regelagent zu Beginn einer Prüfung bei jedem Modellagenten eine Anfrage stellen muss, ist die Anzahl der dabei auszutauschenden Nachrichten proportional zur Anzahl der Modelle und zur Anzahl der Regeln. Die Anzahl der zur eigentlichen Überprüfung einer Regel notwendigen Nachrichten kann nur schwer abgeschätzt werden, da sie davon abhängig ist, wie oft eine Regel angewendet werden muss und wie komplex eine Regel ist bzw. wie viele Informationen sie zur Auswertung benötigt.

Der Prototyp hat im Beispielszenario alle vorhandenen Regeln korrekt angewendet, soweit dies mit den in den Modellen vorhandenen Informationen möglich war. Mit Hilfe der Meta-Meta-Modelle und der lokalen Ontologien ist es also möglich, heterogene Modelle und deren Begriffe auf eine globale Begriffswelt zu übertragen und dort eine Prüfung durchzuführen. Diese kann in bestehende Entwicklungsprozesse integriert werden, ohne diese aufwendig zu ändern, da keine Homogenisierung der dort gewachsenen Modelllandschaft notwendig ist. Ein Benutzer kann eine Prüfung zu einem beliebigen Zeitpunkt anstoßen, da Regeln, die auf Grund des mangelnden Fortschritts in der Entwicklung (nichtausreichende Detaillierung der Modelle), ignoriert werden. Die Ergebnisse einer Prüfung helfen dem Benutzer einerseits Inkonsistenzen zu beheben, andererseits können sie auch aufzeigen, an welchen Stellen eine weitere Detaillierung sinnvoll sein kann. Das System bietet dem Benutzer eine gute Unterstützung bei der Behebung der Inkonsistenzen, in dem alle Informationen, die zur Verletzung einer Regel geführt haben, aufgezeigt werden.

Die globale Ontologie enthält jedoch für die Durchführung des relativ kleinen Beispielszenarios bereits etwa 300 Begriffe, obwohl nur drei Aspekte (Wirkstruktur, Verhalten und Funktionen) abgedeckt werden. Zwar werden sich Begriffe beim Hinzufügen weiterer Aspekte überschneiden und die Ontologie somit nicht linear mit der Anzahl der Aspekte wachsen, jedoch ist die Größe der Ontologie nicht zu vernachlässigen. Eine die gesamte Mechatronik umfassende Ontologie ist nur mit sehr großem Aufwand realisierbar. Die Konzentration sollte deshalb zunächst auf Teilbereichen der Mechatronik, die beispielsweise auf eine spezielle Produktpalette zugeschnitten sind (z.B. Haushaltsgeräte oder Kraftfahrzeuge), liegen. Zu einem späteren Zeitpunkt können diese Ontologien dann erweitert bzw. kombiniert werden.

Eine weitere Herausforderung ist die Wahl der Begriffe der globalen Ontologie und deren Benennung. In der Mechatronik herrscht derzeit keine einheitliche Begriffswelt vor [Thra13]. Um hier eine solide Grundlage zu legen, ist es erforderlich, dass Experten aus unterschiedlichen Bereichen zusammenkommen und diese Aufgabe gemeinsam angehen. Nur wenn das Wissen aller Beteiligten ausreichend einfließen kann, kann eine für die Prüfung heterogener Modelle einsetzbare Ontologie entstehen.

Im Gegensatz dazu beschreiben die lokalen Ontologien lediglich ein Modell bzw. einen Modelltyp und ermöglichen es, diesen in der globalen Begriffswelt zu interpretieren. Im Beispielszenario enthalten die drei lokalen Ontologien jeweils zwischen 50 und 100 Begriffe, können für ei-

nen komplexen Modelltyp jedoch deutlich umfangreicher werden. Sie haben jedoch den Vorteil, dass sie für einen Modelltyp, beispielsweise die Wirkstruktur nach Gausemeier [GFDK09] oder das UML-Zustandsdiagramm, nur einmal erstellt werden müssen und anschließend beliebig oft verwendet werden können. Das ist möglich, da sie einerseits das unveränderliche Meta-Modell (Modellstruktur bzw. Syntax) und andererseits die Bedeutung der einzelnen Modellelemente (Semantik) enthalten. Lediglich wenn im Modell unterschiedliche Bezeichnungen für Modellelemente benutzt werden (z. B. Elektromotor, elektrischer Motor, E-Motor etc.) und diese in der lokalen Ontologie noch nicht hinterlegt sind, müssen diese ergänzt werden.

Das Konzept sowie der entstandene Prototyp sind am Beispiel der Mechatronik entstanden. Prinzipiell ist das Konzept jedoch in allen Bereichen einsetzbar, in dem Systeme von heterogenen, aber auch von homogenen Modellen genutzt werden. Entscheidend ist nur, dass die globale Ontologie alle notwendigen Aspekte des Gebiets beschreibt und die lokalen Ontologien eine Interpretation der Modelle auf globaler Ebene zulassen.

In diesem Kapitel wurde das entwickelte Konzept bzw. der implementierte Prototyp im Rahmen eines Beispielszenarios eingesetzt. Das Szenario besteht aus drei Modellen, die vom Agentensystem auf Konsistenz geprüft wurden. Die Prüfung war, gemessen an den in den Modellen vorhandenen Informationen, erfolgreich und alle vorhandenen Inkonsistenzen wurden aufgedeckt. Jedoch erreichen die Ontologien eine beträchtliche Größe und müssen für umfangreichere Anwendungsfälle, wie sie häufig in der Realität vorkommen, weiter wachsen. Trotzdem ist die Realisierung für abgrenzbare Domänen mit den derzeitigen Mitteln möglich.

Im anschließenden Kapitel folgt eine Zusammenfassung der Arbeit. Es erfolgt eine Bewertung des Konzepts und die Grenzen werden aufgezeigt. Ein Ausblick auf mögliche weiterführende Forschungsarbeiten rundet die Arbeit ab.

# 11 Zusammenfassung und Ausblick

Das in dieser Arbeit entwickelte Konzept ermöglicht die Durchführung einer Prüfung auf einem System heterogener Modelle. Es basiert auf Softwareagenten und semantischen Technologien. Ein Koordinierungsagent interagiert mit dem Benutzer und nimmt einen Prüfungsauftrag von diesem an. Der Auftrag wird an die sogenannten Regelagenten weitergegeben, die jeweils eine Regel repräsentieren und die Modelle auf deren Einhaltung prüfen. Die Modelle werden durch Modellagenten vertreten. Sie stellen die Informationen ihres jeweiligen Modells den Regelagenten einheitlich zur Verfügung und Kapseln damit die Heterogenität der Modelle.

Zu jedem Modell gehört eine Ontologie, mit deren Hilfe Modelle des vorliegenden Typs (z. B. Zustandsdiagramme) interpretiert werden können. Diese Ontologien werden als lokale Ontologien bezeichnet und beschreiben sowohl das Modell als auch wie der Inhalt eines Modells in die globale Begriffswelt übertragen werden kann. Diese wird durch eine globale Ontologie definiert und enthält Begriffe und Zusammenhänge des Fachgebiets, in dem sich die Modelle bewegen (z. B. Mechatronik). Daraus können durch den Ontologieagenten automatisiert Regeln abgeleitet und durch explizit definierte Regeln ergänzt werden.

Am Ende werden die Ergebnisse einer Prüfung an den Koordinierungsagenten zurückgemeldet, der sie für den Benutzer aufbereitet. Diesem werden bei einer aufgedeckten Inkonsistenz alle Informationen bereitgestellt, die zur Verletzung der jeweiligen Regel geführt haben, um ihn bei der Beseitigung der Inkonsistenz zu unterstützen.

## 11.1 Bewertung des Konzepts

Das Konzept zur agentenbasierten Prüfung heterogener Modelle am Beispiel der Mechatronik reduziert die Entwicklungskosten für mechatronische Produkte, da durch Inkonsistenzen zwischen den Modellen hervorgerufene Fehler frühzeitig erkannt und behoben werden können. Zudem wird die Entwicklungszeit verkürzt, da aufwendige manuelle Prüfungen der Modelle entfallen. Ebenso wird die Qualität der entwickelten Produkte gesteigert, indem Fehlerquellen durch ein konsistentes System von Modellen reduziert werden.

Der Benutzer wird bei der Behebung von Problemen unterstützt, indem ihm alle Informationen aus den Modellen bereitgestellt werden, die zur Inkonsistenz geführt haben. Dadurch fällt es ihm leichter, die Inkonsistenzen aufzulösen und er muss weniger Aufwand in die Informationssuche investieren.

Die Prüfung lässt sich zu jedem beliebigen Zeitpunkt im Entwicklungsprozess anwenden, da Regeln, die auf Grund (noch) nicht vorhandener Informationen nicht ausgewertet werden kön-

nen, ignoriert werden. Außerdem können einzelne Regeln deaktiviert werden, um diese dauerhaft zu blockieren. So ist es möglich, Inkonsistenzen gezielt zu ignorieren, um die Kreativität der Entwickler nicht zu beschränken. Zu einem späteren Zeitpunkt können diese dann wieder aktiviert werden und das System der Modelle kann vollständig geprüft werden. Prinzipiell ist auch die Anwendung im Hintergrund (bei jeder Änderung eines Modells durch den Benutzer) möglich, sofern eine ausreichende Rechenleistung gewährleistet ist und der Benutzer durch die Prüfung bei seiner Arbeit nicht aufgehalten wird.

Das Konzept unterstützt explizit den Einsatz von heterogenen Modellen, die mit Hilfe der (lokalen) Ontologien interpretierbar gemacht werden. Die Kapselung der Modelle durch die Modellagenten ermöglicht zudem eine Kopplung heterogener Engineering Werkzeuge, da die Schnittstelle zum Modell bzw. zum Engineering Werkzeug durch den Agenten verwaltet wird. Der häufig gewachsene Entwicklungsprozess muss also nicht neu definiert werden, da bestehende bzw. vertraute Modelle und/oder Engineering Werkzeuge weiterhin genutzt werden können.

Neue Modelle können zur Laufzeit dem System der Modelle hinzugefügt werden. Dieser Umstand wird durch die lose Kopplung der Softwareagenten erreicht. Es müssen lediglich eine entsprechende Schnittstelle zum Modell und eine den Modelltyp beschreibende (lokale) Ontologie vorhanden sein.

Die Technologie der Softwareagenten ermöglicht eine auf mehrere Computer verteilte Ausführung der Prüfung. Die Modellagenten müssen sich also nicht auf demselben Computer befinden. Durch die mehrfach vorhandenen Ressourcen kann damit ein Geschwindigkeitsgewinn erzielt werden. Außerdem ist es möglich, die einzelnen Modelle unterschiedlicher Orte, Abteilungen oder gar Betriebe konsistent zu halten.

Das Konzept ist am Beispiel der Mechatronik entstanden, lässt sich jedoch auch auf andere Domänen übertragen. Entscheidend ist, dass die Modelle mit Hilfe der lokalen Ontologien in die globale Begriffswelt übertragen werden können und diese ausreichend groß ist, um den Inhalt aller Modelle abzudecken. In Kapitel 6.4 wurde gezeigt, dass die Formalisierung prinzipiell auf jedes Modell angewendet werden kann und lediglich der dazu notwendige Aufwand Grenzen setzt. Dasselbe gilt für den Aufwand zur Erstellung der Ontologien.

## 11.2 Notwendige Voraussetzungen und Grenzen

Eine umfassende globale Ontologie ist Voraussetzung zur Anwendung des Konzepts. Sie muss alle Aspekte, die in den einzelnen Modellen abgebildet werden, respektive die dazugehörigen (globalen) Begriffe enthalten. Diese Ontologie muss sehr sorgfältig erstellt werden, da sie die gemeinsame Basis der Modelle und damit der Prüfung der Modelle bildet. Nur wenn Experten aus allen beteiligten Teilgebieten ihre jeweiligen Aspekte sinnvoll miteinander verschmelzen, ist die Ontologie mächtig genug, um eine produktive Anwendung zu ermöglichen. Die Erstellung

und Pflege der Ontologien erfordert jedoch einen erheblichen Aufwand. Um eine breite Anwendung zu erleichtern, sind Systeme, die den Benutzer bei der Verwaltung des Wissens unterstützen, notwendig. In [KRBG11b, KCK+13] ist ein möglicher Ansatz für ein solches Wissensintegrationssystem beschrieben, der an die vorliegende Aufgabenstellung angepasst werden könnte.

Nicht alle notwendigen Regeln können aus den Begriffsrelationen der globalen Ontologie abgeleitet werden. Deshalb ist es notwendig, Regeln manuell zu ergänzen. Dadurch ergibt sich der Vorteil, dass auch individuelle Regeln ergänzt werden können. So lässt sich die Prüfung firmen- und/oder projektspezifisch anpassen.

Wenn die Domäne, in der sich die Modelle bewegen, zu groß ist, so kann der Umfang der globalen Ontologie beträchtlich werden. Dabei kann der Aufwand zur Realisierung so groß werden, dass diese mit derzeitigen Mitteln unmöglich werden kann. Eine vollständige Ontologie „Mechatronik“ oder „Automatisierungstechnik“ wird also in absehbarer Zukunft mit großer Wahrscheinlichkeit nicht existieren und die Realisierung muss deshalb zunächst für einzelne Teilbereiche der Mechatronik angestrebt werden. Es ist also notwendig, die Größe der Domäne entsprechend den verfügbaren Realisierungsmitteln zu wählen.

Im Gegensatz dazu müssen die lokalen Ontologien den von ihnen beschriebenen Modelltyp vollständig beschreiben. Jede Information und jede Aussage eines Modells muss in die globale Begriffswelt übertragen werden können, um eine Prüfung zu ermöglichen. Für einige Modelltypen, die beispielsweise wie die verschiedenen UML-Diagramme eine begrenzte Anzahl an Modellelementtypen und entsprechenden Strukturmustern besitzen, ist dies weniger aufwendig, als zum Beispiel für CAD-Modelle oder andere Modelle mit vielen Freiheitsgraden. Diese besitzen so viele Möglichkeiten in der Modellierung, dass die vollständige Abbildung noch nicht möglich ist (vgl. Kap. 6.4.3). Auch hier gilt, dass die zu prüfenden Modelle entsprechend den zur Verfügung stehenden Realisierungsmitteln auszuwählen sind.

Die Modelle selbst müssen formalisierbar, das heißt in definierte Modellelemente zerlegbar sein, um das Konzept zur Prüfung der Modelle anwenden zu können. Alle im Modell enthaltenen Informationen müssen von einem Rechnersystem ausgelesen und interpretiert werden können. Durch den breiten Einsatz von rechnergestützten Werkzeugen (CAx) ist dies in der Regel gegeben.

Innerhalb der Modelle dürfen nur Begriffe und Bezeichnungen benutzt werden, die in der jeweiligen lokalen Ontologie definiert sind und somit in die globale Begriffswelt übersetzt werden können. Es können also beispielsweise Synonyme für verschiedene Begriffe benutzt werden, sofern diese in der Ontologie angelegt sind. Bei einer freien Begriffswahl bei der Modellierung oder bei Tippfehlern innerhalb der Begriffe, können diese in den Ontologien nicht ermittelt werden. Dies kann zu einer falschen Interpretation der Begriffe führen oder machen diese vollständig unmöglich.

Das vorgestellte Konzept dient lediglich dazu, Inkonsistenzen aufzudecken und den Benutzer bei deren Auflösung durch die Bereitstellung von Informationen zu unterstützen. Eine automatisierte Behebung der Inkonsistenzen oder das Erarbeiten von Lösungsvorschlägen sind nicht Teil des Konzepts.

## 11.3 Ausblick

Die Unterstützung des Benutzers durch die Unterbreitung von Lösungsvorschlägen kann jedoch ein Ziel weiterführender Arbeiten sein. In [KRBG11a] wird eine solche automatisierte Ermittlung von Problemlösungen im Bereich mechanischer Konstruktionen vorgestellt. Diese beschränkt sich allerdings noch auf die Variation von Parametern, die Einfluss auf die Zielgrößen besitzen. Um ganze Strukturen in einem Modell anzupassen, ist die Entwicklung weiterer Konzepte erforderlich. Entsprechendes Problemlösungswissen muss erfasst, formalisiert und integriert werden, welches unter anderem Lösungsprozesse und Entwurfsmuster beinhaltet.

In [DFB11] wird die mangelnde Interoperabilität zwischen Engineering Werkzeugen als ein Hindernis für effizienteres Engineering dargestellt. Das vorliegende Konzept stellt eine mögliche Lösung für die horizontale Interoperabilität (Modelle derselben Engineeringphase) dar. Ein einfaches Beispiel für den Abgleich zweier Modelle in unterschiedlichen Werkzeugen wird in [Arnd13] vorgestellt. Weitere Arbeiten können darauf abzielen, eine von Agenten gekapselte Schnittstelle für Engineering Werkzeuge zu konzipieren, welche dann mit Hilfe einer globalen Ontologie miteinander kommunizieren und Daten bzw. ganze Modelle austauschen könnten.

Diese Idee kann zudem auf die vertikale Interoperabilität (Modelle aufeinander folgender Engineeringphasen) ausgedehnt werden, um die disziplinübergreifende Prüfung in den disziplinspezifischen (späten) Entwurfsphasen der Mechatronik zu ermöglichen. Wenn die Prüfung nicht mehr nur innerhalb einer Entwurfsphase sondern auch zwischen Entwurfsphasen angewendet werden wird, so kann der gesamte Entwurfsprozess bis hin zur Umsetzung überwacht werden.

Weitere Arbeiten können zum Ziel haben, die beispielsweise in [Maur13] vorgestellten Modelltransformationen vollständig automatisiert auszuführen und den Entwicklungsaufwand zu reduzieren. Die Transformationsvorschriften sind dann nicht mehr explizit in einer Bibliothek sondern implizit in einer Ontologie hinterlegt, die für breitere Anwendungsbereiche als bisher ausgelegt werden kann.

In Kombination mit automatisierten Konfigurations- und Planungssystemen, wie sie für Fertigungsmaschinen in [HGB+11, BRHG14] oder für Intralogistiksysteme in [WYG+14] vorgestellt werden, kann eine Weiterentwicklung des vorgestellten Konzepts den Einsatz von heterogenen Komponentenbibliotheken ermöglichen. Die Heterogenität bezüglich der Komponentenbeschreibungen und Parametrierschnittstellen würde dann durch lokale und globale Ontologien überbrückt werden.

Das Konzept kann auch von der Entwicklung von technischen Systemen in deren operativen Betrieb übertragen werden. In Folgearbeiten kann es soweit angepasst werden, dass es beispielsweise auf automatisierungstechnischen Einheiten eingesetzt werden kann. Damit wäre es möglich, auch heterogene Systeme informationstechnisch zu koppeln. So könnten einerseits verschiedene Systeme auf Unternehmensebene gekoppelt werden, wie es in modellbasierter Weise in [HFV13] vorgestellt wird, oder unternehmensübergreifende Systeme miteinander kooperieren, wie in [VDPG14] am Beispiel Cyber-physischer Systeme angedacht. Andererseits ist es jedoch auch denkbar, dass Systeme verschiedener Hersteller auf der Feldebene miteinander kommunizieren, obwohl keine gemeinsame Schnittstelle vorgesehen ist. Ansätze hierzu wird beispielsweise in [Eppl11] oder in [HEMD12] vorgestellt.

# Literaturverzeichnis


[AABL13] H. van der Auweraer, J. Anthonis, S. de Bruyne, J. Leuridan: *Virtual engineering at work: the challenges for designing mechatronic products.* In: Engineering with Computers (2013), Volume 29, Issue 3, S. 389-408. DOI 10.1007/s00366-012-0286-6

[APCB02] C. van Aart, R. Pels, G. Caire, F. Bergenti: *Creating and Using Ontologies in Agent Communication*. In: Proceedings of Workshop on Ontologies in Agent Systems, Bologna, Italy, 2002. http://oas.otago.ac.nz/OAS2002/Papers/oas02-17.pdf Seitenabruf: 02.04.2014

[ARG+13] S. Abele, M. Rauscher, P. Göhner, L. Rodríguez Gómez, H.-J. Wunderlich: *Modellbasierte Generierung von Testdaten für ein eingebettetes System unter Berücksichtigung von Hardware-/Software-Abhängigkeiten mithilfe von Informationsflüssen*. Kongress Mess- und Automatisierungstechnik, Automation 2013, Baden-Baden. In: VDI Berichte 2209, 2013, S. 71-74.

[Arnd13] L. Arndt: *Untersuchung der Einsatzmöglichkeiten von Agenten für heterogene Werkzeugschnittstellen*. Diplomarbeit Nr. 2517, IAS, Universität Stuttgart, 2013.

[Badr11] I. Badr: *Agent-Based Dynamic Scheduling for Flexible Manufacturing Systems*. Dissertation IAS, Universität Stuttgart, 2011. IAS-Forschungsberichte, Bd. 1/2011.

[BGT05] S. Burmester, H. Giese, M. Tichy: *Model-Driven Development of Reconfigurable Mechatronic Systems with MECHATRONIC UML.* In U. Aßmann, M. Aksit, A. Rensink (Hrsg.): Model Driven Architecture: Foundations and Applications. LNCS 3599, Springer-Verlag Berlin Heidelberg, 2005, S. 47-61.

[Bish07] R. Bishop: *The Mechatronics Handbook*. CRC Press, London, England, 2007.

[BJW04] S. Bussmann, N. Jennings, M. Wooldridge: *Multiagent Systems for Manufacturing Control: A Design Methodology*. Springer-Verlag Berlin Heidelberg, 2004.

[Blak13] J. Blake: *Ten Quick Tips for Using the Gene Ontology*. In: PLoS Computational Biology, Volume 9, Issue 11, 2013. doi:10.1371/journal.pcbi.1003343

[BMG08] I. Badr, H. Mubarak, P. Göhner: *Extending the MaSE Methodology for the Development of Embedded Real-Time Systems*. In M. Castani, A. Seghrouchni, J. Leite, P. Torroni (Hrsg.): Languages, Methodologies and Development Tools for Multi-Agent Systems. LNCS 5118, Springer-Verlag Berlin Heidelberg, 2008, S. 106-122.

[BoBo10] P. D. Borches, G. M. Bonnema: *A3 architecture overviews: focusing architectural knowledge to support evolution of complex systems*. In: Proceedings of the 20th Annual International Symposium of INCOSE, Curran Associates, Chicago, USA, 2010.

[BoCh09] J. Borrego-Díaz, A. Chávez-González: *On the Use of Automated Reasoning Systems in Ontology Integration*. In: Proceedings of the Third International Workshop on Ontology, Conceptualization and Epistemology for Information Systems, Software Engineering and Service Science (ONTOSE 2009), 2009, S. 37-48.

[Borc11] P. D. Borches: *A3 Architecture Overviews - Harvesting Architectural Knowledge to Enhance Evolvability of Embedded Systems*. In P. van de Laar, T. Punter (Hrsg.): *Views on Evolvability of Embedded Systems*, Embedded Systems, Springer-Verlag Berlin, Heidelberg, 2011, S. 121-136.

[Bors97] W. Borst: *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. Dissertation CTIT, Universität Twente, Niederlande, 1997. Centre for Telematics and Information Technology, Bd. 14/1997.

[BRHG14] A. Bader, M. Rauscher, U. Heisel, P. Göhner: *Knowledge Based Configuration of Re-configurable Transfer Centres*. In M. Zaeh (Hrsg.): Enabling Manufacturing Competitiveness and Economic Sustainability, Proceedings of the 5th International Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV 2013), Springer International Publishing, 2014, S. 371-376.

[Bröc08] L. Bröcker: *Semiautomatische Erstellung semantischer Netze*. Dissertation Rheinische Friedrich-Wilhelms-Universität Bonn, 2008.
URN: urn:nbn:de:hbz:5N-16381
http://hss.ulb.uni-bonn.de/2008/1638/1638.htm
Seitenabruf: 02.04.2014

[Brus96] H. M. J. Van Brussel: *Mechatronics – A Powerful Concurrent Engineering Framework*. In: IEEE/ASME Transactions on Mechatronics, Volume 1, Issue 2, 1996, S. 127-136.

[BTG04] S. Burmester, M. Tichy, H. Giese: *Modeling Reconfigurable Mechatronic Systems with Mechatronic UML*. In U. Aßmann (Hrsg.): Proceedings of Model Driven Architecture: Foundations and Applications (MDAFA 2004), Linköping, Sweden, S. 155-169, 2004.

[CHF14] L. Christiansen, M. Hoernicke, A. Fay: *Entwicklung eines Modellverbinders zur Vermeidung von Inkonsistenzen zwischen Modellen im Kontext des modellbasierten Engineerings*. In: Kongress, Mess- und Automatisierungstechnik, Automation 2014, VDI Berichte 2231, S. 785-798.

[CJB99] B. Chandrasekaran, J. Josephson, V. Benjamins: *What Are Ontologies, and Why Do We Need Them?* In: IEEE Intelligent Systems and their Applications, Volume 14, Issue 1, 1999, S. 20-26.

[Clus14] Cluster Mechatronik & Automation e.V.: *cluster mechatronic & automation – Mechatronische Organisationsentwicklung*. http://www.cluster-ma.de/themengruppen/mechatr-organisationsentwicklung/index.html Seitenabruf: 26.10.2014

[Come94] R. Comerford: *Mecha...what? [mechatronics]*. IEEE Spectrum, Volume 31, Issue 8, 1994, S.46-49. DOI: 10.1109/6.299539

[Coss05] M. Cossentino: *From Requirements to Code with the PASSI Methodology*. In B. Henderson-Sellers, P. Giorgini (Hrsg.): Agent-Oriented Methodologies, Idea Group Inc., Hershey, PA, USA, 2005, S. 79-106.

[CRR99] M. Clement, G. Rimmel, M. Runte: *Intelligent Software Agents – Implications for Marketing in eCommerce*. In: Proceedings of IRIS 22 (Information Systems Research Seminar in Scandinavia), Jyväskylä, Finland, 1999, S. 203-219.

[DCTK11] K. Dentler, R. Cornet, A. ten Teije, N. de Keizer: *Comparison of Reasoners for large Ontologies in the OWL 2 EL Profile*. In: Semantic Web Journal, Volume 2, Issue 2, 2011, S. 71-87.

[DFB11] R. Drath, A. Fay, M. Barth: *Interoperabilität von Engineering-Werkzeugen*. In: at – Automatisierungstechnik, Band 59, Heft 7, 2011, S. 451-460.

[DFK+04] Y. Ding, D. Fensel, M. Klein, B. Omelayenko, E. Schulten: *The Role of Ontologies in eCommerce*. In S. Staab, R. Studer (Hrsg.): Handbook on Ontologies. Springer-Verlag Berlin Heidelberg, 2004, S. 593-615.

[Ecli14] Eclipse Foundation: *Eclipse*. 2014. http://www.eclipse.org

[EhSt06] M. Ehrig, R. Studer: *Wissensvernetzung durch Ontologien*. In T. Pellegrini, A. Blumauer (Hrsg.): Semantic Web – Wege zur vernetzten Wissensgesellschaft. Springer-Verlag Berlin, Heidelberg, 2006, S. 469-484.

[EKHG01] G. Engels, J. Küster, R. Heckel, L. Groenewegen: *A methodology for specifying and analyzing consistency of object-oriented behavioral models*. In: ESEC/FSE-9 Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, ACM, New York, NY, USA, 2001, S. 186-195.

[EMF14] Eclipse Modeling Framework Project: *Eclipse Modeling Framework*. 2014. http://www.eclipse.org/modeling/emf/

[Eppl11] U. Epple: *Merkmale als Grundlage der Interoperabilität technischer Systeme*. In: at – Automatisierungstechnik, Band 59, Heft 7, 2011, S. 440-450.

[FBRG06] G. de Fombelle, X. Blanc, L. Rioux, M. Gervais: *Finding a Path to Model Consistency*. In A. Rensink, J. Warmer (Hrsg.): ECMDA-FA 2006. LNCS 4066, Springer-Verlag Berlin Heidelberg, 2006, S. 101-112.

[FeGr13] J. Feldhusen, K.-H. Grote (Hrsg.): *Pahl/Beitz Konstruktionslehre - Methoden und Anwendung erfolgreicher Produktentwicklung*. Springer-Verlag Berlin Heidelberg, 2013.

[FGH+94] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, B. Nuseibeh: *Inconsistency Handling in Multiperspective Specifications*. In: IEEE Transactions on Software Engineering, Volume 20, Issue 8, 1994, S. 569-578.

[FGPP99] M. Fernández-López, A. Gómez-Pérez, J. Pazos Sierra, A. Pazos Sierra: *Building a Chemical Ontology Using Methontology and the Ontology Design Environment*. In: IEEE Intelligent Systems and their Applications, Volume 14, Issue 1, 1999, S. 37-46.

[Fink00] A. Finkelstein: *A Foolish Consistency: Technical Challenges in Consistency Management*. In M. Ibrahim, J. Küng, N. Revell (Hrsg.): Database and Expert Systems Applications. Springer-Verlag Berlin Heidelberg, 2000, S. 1-5.

[FIPA02] *FIPA ACL Message Structure Specification*. FIPA (Foundation for Intelligent Physical Agents), 2002.
http://www.fipa.org

[FIPA04] *FIPA Agent Management Specification*. FIPA (Foundation for Intelligent Physical Agents), 2004.
http://www.fipa.org

[FKV14] S. Feldmann, K. Kernschmidt, B. Vogel-Heuser: *Combining a SysML-based Modeling Approach and Semantic Technologies for Analyzing Change Influences in Manufacturing Plant Models*. In: Procedia CIRP, Volume 17, 2014, S. 451-456. http://dx.doi.org/10.1016/j.procir.2014.01.140.

[Fran06] U. Frank: *Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme*. Dissertation Universität Paderborn, Heinz Nixdorf Institut, Rechnerintegrierte Produktion, HNI-Verlagsschriftenreihe, Paderborn, Volume 175, 2006.

[GaKa10] J. Gausemeier, S. Kahl: *Architecture and Design Methodology of Self-Optimizing Mechatronic Systems*. In A. Donato Di Paola and G. Cicirelli (Hrsg.): Mechatronic Systems Simulation Modeling and Control, InTech, 2010, S. 255-282. DOI: 10.5772/9121.

[GFDK09] J. Gausemeier, U. Frank, J. Donoth, S. Kahl: *Specification technique for the description of self-optimizing mechatronic systems*. In: Research in Engineering Design, Volume 20, Issue 4, 2009, S. 201-223.

[GFS05] J. Gausemeier, U. Frank, B. Schulz: *Domänenübergreifende Spezifikation der Prinziplösung selbstoptimierender Systeme unter Berücksichtigung der auf das System wirkenden Einflüsse*. In: Mechatronik 2005 - Innovative Produktentwicklung, VDI Berichte 1892.1, 2005, S. 315-336.

[GGB11] L. García, A. García, J. Bateman: *An Ontology-Based Feature Recognition and Design Rule Checker for Engineering*. In K. Baclawski, J. Bateman, A. García Castro,C. Lange, K. Viljanen (Hrsg.): Proceedings of the Workshop "Ontologies come of Age in the Semantic Web" (OCAS2011), 10th International Semantic Web Conference, Bonn, 2011, S. 48-59.
http://ceur-ws.org/Vol-809/paper-07.pdf
Seitenabruf: 22.05.2014

[GGS+07] J. Gausemeier, H. Giese, W. Schäfer, B. Axenath, U. Frank, S. Henkler, S. Pook, M. Tichy: *Towards the Design of Self-Optimizing Mechatronic Systems: Consistency Between Domain-Spanning and Domain-Specific Models*. In: J.-C. Bocquet (Hrsg.): Proceedings of the 16th International Conference on Engineering Design (ICED07). Paris, Frankreich, 2007, S. 657-668.

[GKB13] G. Grimnes, M. Kiesel, A. Bernardi: *Ontology-Based Mobile Communication in Agriculture*. In: KI - Künstliche Intelligenz, Volume 27, Issue 4, 2013, S. 335-339.

[GMF14] Graphical Modeling Project: *Graphical Modeling Framework*. 2014.
http://www.eclipse.org/modeling/gmp/

[Göhn13] P. Göhner (Hrsg.): *Agentensysteme in der Automatisierungstechnik*. Reihe: Xpert.press, Springer-Verlag Berlin Heidelberg, 2013.

[Göhn14] P. Göhner: *Vorlesung Softwaretechnik II*. IAS, Universität Stuttgart, Sommersemester 2014.

[Gram11] M. Gramm: *Formalisierung der Modelle des mechatronischen Entwurfs und Implementierung eines graphischen Editors*. Diplomarbeit Nr. 2371, IAS, Universität Stuttgart, 2011.

[Grub93] T. Gruber: *A Translation Approach to Portable Ontology Specifications*. In: Knowledge Acquisition 5(2), 1993, S. 199-220.

[GSG+09] J. Gausemeier, W. Schäfer, J. Greenyer, S. Kahl, S. Pook, J. Rieke: *Management of Cross-Domain Model Consistency during the Development of Advanced Mechatronic Systems*. In M. Norell Bergendahl, M. Grimheden, L. Leifer, P. Skogstad, U. Lindemann (Hrsg.): Proceedings of the 17th International Conference on Engineering Design (ICED 09), Volume 6, Design Methods and Tools (pt. 2), Palo Alto, CA, USA, 2009, S. 1-12.

[GUW04] P. Göhner, P. de A. Urbano, T. Wagner: *Softwareagenten - Einführung und Überblick über eine alternative Art der Softwareentwicklung. Teil III: Agentensysteme in der Automatisierungstechnik: Aufbau, Struktur und Implementierung an einem Anwendungsbeispiel*. In: Automatisierungstechnische Praxis 46 (2004), H. 2, S. 42-51.

[HEMD12] T. Hadlich, C. Engel, M. Mühlhause, C. Diedrich: *Konzept und Erfahrungen beim Abgleich mehrerer Domain-Ontologien*. In U. Jumar, E. Schnieder, C. Diedrich (Hrsg.): Entwurf komplexer Automatisierungssysteme EKA 2012, Magdeburg, 2012, S. 81-90.

[Hepp08] M. Hepp: *Good Relations: An Ontology for Describing Products and Services Offers on the Web*. In A. Gangemi, J. Euzenat (Hrsg.): Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW) 2008, LNCS 5268, Springer-Verlag Berlin Heidelberg, 2008, S. 329-346.

[Herr11] H. Herre: *The Ontology of Medical Terminological Systems – Towards the Next Generation of Biomedical Ontologies*. In R. Poli, J. Seibt, M. Healy, A. Kameas (Hrsg.): Theory and Applications of Ontology. Volume 2, Springer Netherlands, 2011, S. 373-391.

[Hess14] W. Hesse: *Ontologie(n)*. In: Informatiklexikon, Gesellschaft für Informatik (GI).
http://www.gi.de/service/informatiklexikon/detailansicht/article/ontologien.html
Seitenabruf: 28.03.2014

[HEZ10] P. Hehenberger, A. Egyed, K. Zeman: *Consistency checking of mechatronic design models*. In: Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2010), ASME, Volume 3, 2010, S. 1141-1148.

[HEZ12] P. Hehenberger, A. Egyed, K. Zeman: *Understanding the Relationship of Information in Mechatronic Design Modeling*. In R. Moreno-Díaz, F. Pichler, A. Quesada-Arencibia (Hrsg.): Computer Aided Systems Theory – EUROCAST 2011, 13th International Conference EUROCAST 2011, Volume 2, Springer Berlin Heidelberg, 2012, S. 113-120.

[HFV13] J. Hufnagel, T. Frank, B. Vogel-Heuser: *Framework for a Model-Based, Cross-Domain System Interconnection in Automation Technology*. In: 18th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Cagliari, Italien, 2013, S. 1-9.

[HGB+10] U. Heisel, P. Göhner, A. Bader, W. Maier, C. Malz, M. Rauscher: *Transferzentren intelligent (re-)konfigurieren – Effizienzsteigerung in der Produktion durch agentenbasierte Fertigungssystemplanung*. In: wt Werkstattstechnik online, Jahrgang 100 (2010), H. 7/8, 2010, S. 553-558.

[HGB+11] U. Heisel, P. Göhner, A. Bader, M. Rauscher, W. Schulleri: *Flexible Fertigungssystemplanung für Transferzentren – Agentenorientiertes Modell zur Planung rekonfigurierbarer Fertigungssysteme*. In: wt Werkstattstechnik online, Jahrgang 101 (2011), H. 4, 2011, S. 200-205.

[HGBR14] U. Heisel, P. Göhner, A. Bader, M. Rauscher: *Effektivität in der Powertrain-Fertigung steigern - Agentenorientiertes Softwaresystem zur Rekonfiguration modularer Transferzentren*. In: wt Werkstattstechnik online, Jahrgang 104 (2014), H. 1/2, 2014, S. 89-96.

[HPH03] I. Horrocks, P. Patel-Schneider, F. Van Harmelen: *From SHIQ and RDF to OWL: the making of a Web Ontology Language*. In: Journal of Web Semantics, Volume 1, Issue 1, 2003, S. 7-26.

[HQRP11] S. Herzig, A. Qamar, A. Reichwein, C. Paredis: *A Conceptual Framework for Consistency Management in Model-Based Systems Engineering*. In: Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011, ASME, 2011, S. 1329-1339.

[HSM+02] J. Hirtz, R. Stone, D. McAdams, S. Szykman, K. Wood: *A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts*. NIST (National Institute of Standards and Technology) Technical Note 1447, Gaithersburg, USA, 2002.

[Iser99] R. Isermann: *Mechatronische Systeme: Grundlagen*. Springer-Verlag Berlin Heidelberg, 1999.

[JADE10] *JADE Programmer's Guide (JADE 4.0)*. F. Bellifemine, G. Caire, T. Trucco, G. Rimassa, TILab S.p.A., 2010.
http://jade.tilab.com/

[Jans10] K. Janschek: *Systementwurf mechatronischer Systeme*, Springer-Verlag Berlin Heidelberg, 2010.

[JBR+12] M. Jüttner, A. Buchau, M. Rauscher, W. Rucker, P. Göhner: *Software Agent Based Domain Decomposition Method*. In: Proceedings of the 15th International IGTE Symposium on Numerical Field Calculation in Electrical Engineering (IGTE'12), Graz, Austria, 2012, S. 89-94.

[JBR+13] M. Jüttner, A. Buchau, M. Rauscher, W. Rucker, P. Göhner: *Iterative Solution of Multiphysics Problems with Software Agents Designed as Physics Experts*. In I. Dolezel et al. (Hrsg.): Proceedings of the International Symposium on Theoretical Electrical Engineering (ISTET 2013). Pilsen, Tschechien, 2013.

[Jenn00] N. Jennings: *On agent-based software engineering*. In: Artificial Intelligence 117 (2000), 2000, S. 277-296.

[JeWo01] N. Jennings, M. Wooldridge: *Agent-oriented software engineering*. In J. Bradshaw (Hrsg.): Handbook of Agent Technology, AAAI/MIT Press, 2001.
http://users.ecs.soton.ac.uk/nrj/download-files/agt-handbook.pdf
Seitenabruf: 27.06.2014

[KCK+13] M. Kratzer, A. Crostack, R. Kadadihi, M. Rauscher, H. Binz, P. Göhner: *A concept of direct knowledge acquisition for multi-agent design systems*. In U. Lindemann, S. Venkataraman, Y. Kim, S. Lee, J. Clarkson, G. Cascini (Hrsg.): Proceedings of the 19th International Conference on Engineering Design (ICED13), Volume 6, 2013, S. 219-228.

[Keet13] M. Keet: *Open World Assumption*. In W. Dubitzky, O. Wolkenhauer, K. Cho, H. Yokota (Hrsg.): Encyclopedia of Systems Biology, Springer New York, 2013, S. 1567.

[KeLe12] M. Keshavarz, Y. Lee: *Ontology matching by using ConceptNet*. In V. Kachitvichyanukul, H.T. Luong, R. Pitakaso (Hrsg.): Proceedings of the Asia Pacific Industrial Engineering & Management Systems Conference 2012, 2012, S. 1917-1925.
http://www.apiems.net/conf2012/
Seitenabruf: 27.06.2014

[KoKa94] R. Koller, N. Kastrup: *Prinziplösungen zur Konstruktion technischer Produkte*. Springer-Verlag Berlin Heidelberg, 1994.

[Koll98] R. Koller: *Konstruktionslehre für den Maschinenbau*. Springer-Verlag Berlin Heidelberg, 1998.

[KRBG11a] M. Kratzer, M. Rauscher, H. Binz, P. Göhner: *An agent-based system for supporting design engineers in the embodiment design phase*. In: S. Culley, B. Hicks, T. Mcaloone, T. Howard, A. Dong (Hrsg.): Proceedings of the 18th International Conference on Engineering Design (ICED11). Kopenhagen, Dänemark, 2011, S. 178-189.

[KRBG11b] M. Kratzer, M. Rauscher, H. Binz, P. Göhner: *Konzept eines Wissensintegrationssystems zur benutzerfreundlichen, benutzerspezifischen und selbstständigen Integration von Konstruktionswissen*. In D. Krause, K. Paetzold, S. Wartzack (Hrsg.): Proceedings of the 22nd Symposium Design for X. Tutzing, 2011, S. 21-32.

[KuGe13] W. Kugler, D. Gehlich: *Einsatz von Agentensystemen in der Intralogistik*. In P. Göhner (Hrsg.): Agentensysteme in der Automatisierungstechnik. Reihe: Xpert.press, Springer-Verlag Berlin Heidelberg, 2013, S. 113-128.

[LaGö99] R. J. Lauber, P. Göhner: *Prozessautomatisierung 1*. 3. Aufl., Springer-Verlag Berlin Heidelberg, 1999.

[Lee10] J. Lee: *Automated checking of building requirements on circulation over a range of design phases*. Dissertation Department Architecture, Georgia Institute of Technology, USA, 2010.
http://hdl.handle.net/1853/34802
Seitenabruf: 24.04.2014

[LMB09] U. Lindemann, M. Maurer, T. Braun: *Structural Complexity Management*. Springer-Verlag Berlin Heidelberg, 2009.

[LSLF14] C. Legat, C. Seitz, S. Lamparter, S. Feldmann: *Semantics to the Shop Floor: Towards Ontology Modularization and Reuse in the Automation Domain*. In: Preprints of the 19th IFAC World Congress, Kapstadt, Südafrika, 2014, S. 3444-3449.

[Malz13] C. Malz: *Agentenbasierte dynamische Testfallpriorisierung*. Dissertation IAS, Universität Stuttgart, 2013. IAS-Forschungsberichte, Bd. 2/2013.

[MaNo10] C. Macal, M. North: *Tutorial on agent-based modelling and simulation*. In: Journal of Simulation (2010) 4/3, 2010, S. 151-162.

[Maur13] M. Maurmaier: *Modellgetriebene Entwicklung von Automatisierungssystemen*. Dissertation IAS, Universität Stuttgart, 2013. IAS-Forschungsberichte, Bd. 1/2013.

[MMWY10] W. Marquardt, J. Morbach, A. Wiesner, A. Yang: *OntoCAPE – A Re-Usable Ontology for Chemical Process Engineering*. Springer-Verlag Berlin Heidelberg, 2010.

[Moeh09] S. Moehringer: *Mechatronic Design – Historical background and challenges for the future*. In M. Norell Bergendahl, M. Grimheden, L. Leifer, P. Skogstad, U. Lindemann (Hrsg.): Proceedings of the 17th International Conference on Engineering Design (ICED 09), Volume 4, Product and Systems Design, Palo Alto, CA, USA, 2009, S. 333-344.

[Nerd11] F. Nerdinger: *Teamarbeit*. In F. Nerdinger, G. Blickle u. N. Schaper (Hrsg.): Arbeits- und Organisationspsychologie. Springer-Verlag Berlin Heidelberg, 2011, S. 95-109. DOI 10.1007/978-3-642-16972-4_8

[NSH+08] R. Nagel, R. Stone, R. Hutcheson, D. McAdams, J. Donndelinger: *Function Design Framework (FDF): Integrated Process and Function Modeling for Complex Systems*. In: Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2008, Brooklyn, New York, USA, 2008, S. 273-286. IDETC2008- 49369.

[OCM+07] L. Obrst, W. Ceusters, I. Mani, S. Ray, B. Smith: *The evaluation of ontologies*. In C. Baker, K. Cheung (Hrsg.): Revolutionizing Knowledge Discovery in the Life Sciences, Springer-Verlag Berlin Heidelberg, 2007, S. 139-158.

[OMG11] *OMG Unified Modeling Language (OMG UML), Infrastructure*. Object Management Group, 2011.
http://www.omg.org/spec/UML/2.4.1/Infrastructure

[OMG12] *OMG Systems Modeling Language (OMG SysML)*. Object Management Group, 2012.
http://www.omg.org/spec/SysML/1.3/

[OMG14] *OMG Meta Object Facility (MOF) Core Specification*. Object Management Group, 2014.
http://www.omg.org/spec/MOF/2.4.2

[Pech14] S. Pech: *Agentenbasierte Informationsgewinnung für automatisierte Systeme*. Dissertation IAS, Universität Stuttgart, 2014. IAS-Forschungsberichte, Bd. 1/2014.

[PeGö10] S. Pech, P. Göhner: *Flexible Industrial Automation and Control Systems Based on Qualitative Models and Software Agents*. In: 22nd International Conference on System Research, Informatics & Cybernetics, Baden-Baden, 2010.

[RaGö12a] M. Rauscher, P. Göhner: *Konsistenzprüfung im frühen mechatronischen Entwurf*. In U. Jumar, E. Schnieder, C. Diedrich (Hrsg.): Entwurf komplexer Automatisierungssysteme EKA 2012, Magdeburg, 2012, S. 199-207.

[RaGö12b] M. Rauscher, P. Göhner: *Automated Consistency Check in early Mechatronic Design*. In R. Scheidl, B. Jakoby (Hrsg.): The 13th Mechatronics Forum International Conference (Mechatronics 2012), Proceedings Vol. 3/3, 2012, S. 800-803.

[RaGö13a] M. Rauscher, P. Göhner: *Konsistenzprüfung im frühen mechatronischen Entwurf*. In: at – Automatisierungstechnik, Band 61, Heft 2, 2013, S. 109-113.

[RaGö13b] M. Rauscher, P. Göhner: *Agent-based consistency check in early mechatronic design phase*. In U. Lindemann, S. V, Y. Kim, S. Lee, J. Clarkson, G. Cascini (Hrsg.): Proceedings of the 19th International Conference on Engineering Design (ICED13), Vol. 9, 2013, S. 389-396.

[RFSE11] S. Runde, A. Fay, S. Schmitz, U. Epple: *Wissensbasierte Systeme im Engineering der Automatisierungstechnik*. In: at – Automatisierungstechnik 59 (2011) 1, S. 42-49.

[RJG11] M. Rauscher, N. Jazdi, P. Göhner: *Erfahrungen bei der Entwicklung eines mechatronischen Systems – der automatisierte Fußballschuh David*. In J. Gausemeier, F. Rammig, W. Schäfer, A. Trächtler (Hrsg.): 8. Paderborner Workshop Entwurf mechatronischer Systeme, HNI-Verlagsschriftenreihe Band 294, 2011, S. 289-301.

[RPKC11] C. Roussey, F. Pinet, M. Ah Kang, O. Corcho: *An Introduction to Ontologies and Ontology Engineering*. In G. Falquet, C. Métral, J. Teller, C. Tweed (Hsrg.): Ontologies in Urban Development Projects. Springer-Verlag Berlin Heidelberg, 2011, S. 9-38.

[RSMG11] M. Rauscher, W. Schulleri, C. Malz, P. Göhner: *Agentenbasierte Fertigungssystemplanung für den Einsatz von Transferzentren*. In: Kongress, Mess- und Automatisierungstechnik, Automation 2011, VDI Berichte 2143, S. 267-270.

[RSRV14] D. Regulin, M. Schneider, S. Rehberger, B. Vogel-Heuser: *Automated model generation in the Field of electrical automotive driveline components*. In: Preprints of the 19th IFAC World Congress, Kapstadt, Südafrika, 2014, S. 4499-4504.

[SaAl08] A. Salem, M. Alfonse: *Ontology versus Semantic Networks for Medical Knowledge Representation*. In N. Mastorakis, V. Mladenov, Z. Bojkovic, D. Simian, S. Kartalopoulos, A. Varonides, C. Udriste, E. Kindler, S. Narayanan, J. Mauri, H. Parsiani, K. Man (Hrsg.): New Aspects of Computers – Proceedings of the 12th WSEAS International Conference on COMPUTERS, WSEAS Press, 2008, S. 769-774.

[Schü95] A. Schürr: *Specification of graph translators with triple graph grammars*. In E. Mayr, G. Schmidt, G. Tinhofer (Hrsg.): Graph-Theoretic Concepts in Computer Science, Springer Berlin Heidelberg, 1995, S. 151-163.

[SCR08] E. Salas, N. Cooke, M. Rosen: *On Teams, Teamwork, and Team Performance: Discoveries and Developments.* In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* (2008), Volume 50 No. 3, S. 540-547. DOI 10.1518/001872008X288457

[SCT05] C. Silva, J. Castro and P. Tedesco: *Using Environment Properties to Select Agent Architectures*. In: Proceedings of SEAS 2005 – First Workshop on Software Engineering for Agent-oriented Systems, 19th Brazilian Symposium on Software Engineering (XIX SBES), Uberlância, Brazil, 2005, S. 69-74.

[SGF12] M. Suárez-Figueroa, A. Gómez-Pérez, M. Fernández-López: *The NeOn Methodology for Ontology Engineering*. In M. Suárez-Figueroa, A. Gómez-Pérez, E. Motta, A. Gangemi (Hrsg.): Ontology Engineering in a Networked World. Springer-Verlag Berlin Heidelberg, 2012, S. 9-34.

[SKM01] T. Schäfer, A. Knapp, S. Merz: *Model Checking UML State Machines and Collaborations*. In: Electronic Notes in Theoretical Computer Science, Volume 55, Issue 3, 2001, S. 357-369.

[SMSJ03] R. van der Straeten, T. Mens, J. Simmonds, V. Jonckers: *Using Description Logic to Maintain Consistency between UML Models*. In P. Stevens, J. Whittle, G. Booch: «UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications, Springer-Verlag Berlin Heidelberg, 2003, S. 326-340.

[SSS09] Y. Sure, S. Staab, R. Studer: *Ontology Engineering Methodology*. In S. Staab, R. Studer (Hrsg.): Handbook on Ontologies, International Handbooks on Information Systems, Springer-Verlag Berlin Heidelberg, 2009, S. 135-152.

[Stan14] Stanford Center for Biomedical Informatics Research: *protégé*. 2014. http://protege.stanford.edu/

[SuBu01] K. Sundermeyer, S. Bussmann: *Einführung der Agententechnologie in einem produzierenden Unternehmen – Ein Erfahrungsbericht*, Wirtschaftsinformatik 43, Nr. 2, 2001, S. 135-142.

[Thra13] K. Thramboulidis: *Overcoming Mechatronic Design Challenges: the 3+1 SysML-view Model.* In: Computing Science and Technology International Journal (2013), Volume 1, Issue 1, S. 6-14.

[VDI2206] VDI 2206: *Entwicklungsmethodik für mechatronische Systeme.* Verein Deutscher Ingenieure, Beuth Verlag, Berlin, 2004.

[VDI2221] VDI 2221: *Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte.* Verein Deutscher Ingenieure, Beuth Verlag, Berlin, 1993.

[VDI2653a] VDI 2653 Blatt 1: *Agentensysteme in der Automatisierungstechnik – Grundlagen*. Verein Deutscher Ingenieure, Beuth Verlag, Berlin, 2010.

[VDI2653b] VDI 2653 Blatt 2: *Agentensysteme in der Automatisierungstechnik – Entwicklung*. Verein Deutscher Ingenieure, Beuth Verlag, Berlin, 2012.

[VDI2653c] VDI 2653 Blatt 3: *Agentensysteme in der Automatisierungstechnik – Anwendung*. Verein Deutscher Ingenieure, Beuth Verlag, Berlin, 2012.

[VDPG14] B. Vogel-Heuser, C. Diedrich, D. Pantförder, P. Göhner: *Coupling heterogeneous production systems by a multi-agent based cyber-physical production system*. In: 12th IEEE International Conference on Industrial Informatics (INDIN), Porto Alegre, Brasilien, 2014, S. 713-719.

[Vran09] D. Vrandecic: *Ontology Evaluation*. In S. Staab, R. Studer (Hrsg.): Handbook on Ontologies, International Handbooks on Information Systems, Springer-Verlag Berlin Heidelberg, 2009, S. 293-314.

[VSFL14] B. Vogel-Heuser, D. Schütz, T. Frank, C. Legat: *Model-driven engineering of Manufacturing Automation Software Projects – A SysML-based approach*. In: Mechatronics, Volume 24, Issue 7, 2014. S. 883-897. http://dx.doi.org/10.1016/j.mechatronics.2014.05.003

[W3C12] *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation, World Wide Web Consortium, 2012. http://www.w3.org/TR/owl2-syntax/

[W3C13] *SPARQL 1.1 Query Language*. W3C Recommendation, World Wide Web Consortium, 2013. http://www.w3.org/TR/sparql11-query/

[W3C14] *Resource Description Framework (RDF) 1.1 Semantics*. W3C Recommendation, World Wide Web Consortium, 2014. http://www.w3.org/TR/rdf11-mt/

[Wagn08] T. Wagner: *Agentenunterstütztes Engineering von Automatisierungsanlagen*. Dissertation IAS, Universität Stuttgart, 2008. IAS-Forschungsberichte, Bd. 1/2008.

[Wann10] A. Wannagat: *Entwicklung und Evaluation agentenorientierter Automatisierungssysteme zur Erhöhung der Flexibilität und Zuverlässigkeit von Produktionsanlagen*. Dissertation AIS, Technische Universität München, Sierke-Verlag, 2010.

[Weiß01] G. Weiß: *Agent Orientation in Software Engineering*. In: The Knowledge Engineering Review, Volume 16, Issue 04, 2001, S. 349-373.

[WGU03] T. Wagner, P. Göhner, P. de A. Urbano: *Softwareagenten - Einführung und Überblick über eine alternative Art der Softwareentwicklung. Teil I: Agentenorientierte Softwareentwicklung*. In: Automatisierungstechnische Praxis 45 (2003), H. 10, S. 48-57.

[WJK00] M. Wooldridge, N. Jennings, D. Kinny: *The Gaia Methodology for Agent-Oriented Analysis and Design*. Journal of Autonomous Agents and Multi-Agent Systems. 3, 3 (2000), 2000, S. 285-312.

[WLT+06] K. Wolstencroft, P. Lord, L. Tabernero, A. Brass, R. Stevens: *Protein classification using ontology classification*. In: Bioinformatics, Volume 22, Issue 14, 2006, S. 530-538.

[WoJe95] M. Wooldridge, N. Jennings: *Agent theories, architectures, and languages: A survey*. In M. Wooldridge, N. Jennings (Hrsg.): Intelligent Agents, Proceedings of the ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag Berlin Heidelberg, 1995, S. 1-39.

[WoLe01] M. Wood, S. DeLoach: *An Overview of the Multiagent Systems Engineering Methodology*. In P. Ciancarini, M. Wooldridge (Hrsg.): Proceedings of the First International Workshop on Agent-Oriented Software Engineering, Limerick, Ireland. LNCS 1957, Springer-Verlag Berlin Heidelberg, 2001, S. 207-221.

[WYG+14] K.-H. Wehking, R. Yousefifar, P. Göhner, M. Bordasch, M. Rauscher: *Agentenbasierte Planung von Intralogistiksystemen*. In: Jahrbuch Logistik 2014.

[Zhan14] Haifei Zhang: *A Query Driven Method of Mapping from Global Ontology to Local Ontology in Ontology-based Data Integration*. In: Journal of Software, Volume 9, Issue 3, 2014, S. 738-742.

[ZLS05] W.J. Zhang, Y. Lin, N. Sinhal: *On the Function-Behavior-Structure Model for Design*. In: Proceedings of the Canadian Design Engineering Network (CDEN) Conference, Kaninaskis, Kanada, 2005.
http://library.queensu.ca/ojs/index.php/PCEEA/issue/view/397
Seitenabruf: 27.06.2014

[ZTBD02] W.Y. Zhang, S.B. Tor, G.A. Britton, Y.M. Deng: *Functional design of mechanical products based on behavior driven function-environment-structure modeling framework*. In: Innovation in Manufacturing Systems and Technology (IMST), 2002.
http://hdl.handle.net/1721.1/4031
Seitenabruf: 30.01.2014

# Anhang A: Ausschnitte aus den Ontologien

Die globale Ontologie des Beispielszenarios besteht aus etwa 300 Begriffen und ist in OWL formuliert. Im Folgenden sind verschiedene Ausschnitte der globalen Ontologie dargestellt.

```
<?xml version="1.0"?>

<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<Ontology xmlns="http://www.w3.org/2002/07/owl#"

xml:base="http://www.semanticweb.org/michaelrauscher/ontologies/globalOntology"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     ontologyIRI=
         "http://www.semanticweb.org/michaelrauscher/ontologies/globalOntology">
    <Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Import>
        http://www.semanticweb.org/michaelrauscher/ontologies/consistencyRule
    </Import>
    <Import>
        http://www.semanticweb.org/michaelrauscher/ontologies/systemstructure
    </Import>
    <Import>
        http://www.semanticweb.org/michaelrauscher/ontologies/statechart
    </Import>
    <Declaration>
        <Class IRI="#Aktion"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Asynchronmotor"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Dimension"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Einheit"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Elektrischer_Motor"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Elementarfunktion"/>
    </Declaration>
...
```

```
...
    <SubClassOf>
        <Class IRI="#Asynchronmotor"/>
        <Class IRI="#Elektrischer_Motor"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Elektrischer_Motor"/>
        <Class IRI="#Motor"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Elektrischer_Motor"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#verbraucht"/>
            <Class IRI="#elektrische_Energie"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Energiefluss"/>
        <Class IRI="#Fluss"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Energiequelle"/>
        <Class IRI="#Quelle"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Energiequelle"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#produziert"/>
            <Class IRI="#Energie"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Energiesenke"/>
        <Class IRI="#Senke"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Energiesenke"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#verbraucht"/>
            <Class IRI="#Energie"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Gleichstrommotor"/>
        <Class IRI="#Elektrischer_Motor"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Groesse"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hat"/>
            <Class IRI="#Dimension"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Groesse"/>
        <ObjectSomeValuesFrom>
            <ObjectProperty IRI="#hat"/>
            <Class IRI="#Einheit"/>
        </ObjectSomeValuesFrom>
    </SubClassOf>
...
```

Für jeden Modelltyp muss eine lokale Ontologie als Beschreibung vorhanden sein. Im Folgenden ist ein Ausschnitt der lokalen Ontologie zum Modell des Zustandsdiagramms dargestellt. Vor den Deklarationen der Begriffe, befindet sich das Import-Statement (<Import>) für die globale Ontologie. Dies ist erforderlich, damit lokale Begriffe mit der globalen Begriffswelt verknüpft werden können. Nach den Deklarationen sind Beispiele solcher Verknüpfungen dargestellt.

```
...
<Ontology xmlns="http://www.w3.org/2002/07/owl#"

xml:base="http://www.semanticweb.org/michaelrauscher/ontologies/localOntology1"
...
     ontologyIRI=
         "http://www.semanticweb.org/michaelrauscher/ontologies/localOntology1">
...
    <Import>
        http://www.semanticweb.org/michaelrauscher/ontologies/globalOntology
    </Import>
    <Declaration>
        <Class IRI="#Do-Aktion"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Entry-Aktion"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Exit-Aktion"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Transition"/>
    </Declaration>
    <Declaration>
        <Class IRI="#Zustand"/>
    </Declaration>
    <SubClassOf>
        <Class IRI="#Do-Aktion"/>
        <Class IRI=
"http://www.semanticweb.org/michaelrauscher/ontologies/globalOntology#Aktion"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Entry-Aktion"/>
        <Class IRI=
"http://www.semanticweb.org/michaelrauscher/ontologies/globalOntology#Aktion"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Exit-Aktion"/>
        <Class IRI=
"http://www.semanticweb.org/michaelrauscher/ontologies/globalOntology#Aktion"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Transition"/>
        <Class IRI=
"http://www.semanticweb.org/michaelrauscher/ontologies/statechart#Logischer_Zust
andsübergang"/>
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Zustand"/>
        <Class IRI=
"http://www.semanticweb.org/michaelrauscher/ontologies/statechart#Logischer_Zust
and"/>
    </SubClassOf>
...
```

# Lebenslauf

### Persönliche Daten

| | |
|---|---|
| 15.12.1982 | geboren in Stuttgart – Bad Cannstatt |

### Schulbildung

| | |
|---|---|
| 1989 – 1993 | Mörikeschule (Grundschule), Backnang |
| 1993 – 2002 | Gymnasium in der Taus, Backnang, Abschluss Abitur |

### Zivildienst

| | |
|---|---|
| 2002 – 2003 | Backnanger Werkstätten der Paulinenpflege Winnenden<br>(Werkstatt für geistig und körperlich behinderte Menschen) |

### Studium

| | |
|---|---|
| 2003 – 2008 | Studium der Elektrotechnik und Informationstechnik an der Universität Stuttgart<br>Studienschwerpunkt: Automatisierungs- und Regelungstechnik<br>Vertiefungsrichtung: Automatisierungs- und Softwaretechnik<br>Abschluss: Diplom-Ingenieur |
| | Nebentätigkeit als PC-Dienstleister als Selbstständiger in Backnang |
| 22.10.2008 | Abschluss als Diplom-Ingenieur |

### Berufstätigkeit

| | |
|---|---|
| November 2008 – Oktober 2013 | Wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart |
| Dezember 2011 – Juni 2015 | Projektleiter der TGU Tonsäulen |
| seit September 2014 | Automatisierungsingenieur bei der Harro Höfliger Verpackungsmaschinen GmbH |