Institut für Informatik
Softwaretechnik
Warburger Str. 100
33098 Paderborn

Entwurf mechatronischer Systeme auf Basis von Funktionshierarchien und Systemstrukturen

Schriftliche Arbeit
zur Erlangung des Grades
“Doktor der Naturwissenschaften“

vorgelegt von

Dipl.-Wirt. Inform. Matthias Gehrke
Fürstenallee 39
33102 Paderborn

Paderborn, Oktober 2005

Zusammenfassung

„Software ist überall“. Diese Tatsache trifft mittlerweile auf viele Bereiche des täglichen Lebens zu. Software befindet sich in Mobiltelefonen, Kühlschränken, Kaffeemaschinen, im Toaster, in Flugzeugen oder in Kraftfahrzeugen. In vielen Fällen ist man sich ihrer Präsenz jedoch gar nicht bewusst. Im Falle des Anti-Blockier-Systems (ABS) zum Beispiel, sieht der Fahrer eines Kraftfahrzeuges gar nicht, dass der Bremsvorgang von einem Softwaresystem überwacht und gesteuert wird. Bestimmte Sensoren messen dabei die Drehzahl der Räder und melden dies dem Softwaresystem. Dieses steuert, im Falle eines Blockierens, den jeweiligen Bremszylinder so an, dass dieser weniger stark auf die Bremsscheibe drückt und das blockierte Rad wieder frei gibt. Diese Art von Systemen, bei denen die Elemente sowohl aus der Informationstechnik, als auch aus der Elektrotechnik und dem Maschinenbau stammen, werden *Mechatronische Systeme* genannt.

Die Entwicklung solcher mechatronischen Systeme stellt die Entwicklerteams heutzutage vor große Herausforderungen. Alle Vorteile, Nachteile und Wechselwirkungen der verschiedenen Disziplinen müssen berücksichtigt werden, um ein optimales Zusammenwirken zu erreichen. Nur wenn alle Disziplinen bereits sehr früh während des Systementwurfs berücksichtigt werden, können qualitativ hochwertige mechatronische Systeme entwickelt werden.

Während des Systementwurfs werden zur Beschreibung unterschiedlicher Sachverhalte, wie z.B. der Anforderungen oder der Struktur und das Verhalten der verwendeten Elemente, unterschiedliche Spezifikationstechniken eingesetzt. Viele dieser Spezifikationstechniken hängen voneinander ab bzw. bauen aufeinander auf. Zwei dieser Spezifikationstechniken sind die *Funktionshierarchie* und die *Systemstruktur*. Die Funktionshierarchie wird dazu verwendet, die Funktionen des zu entwickelnden mechatronischen Systems zu beschreiben und in der Systemstruktur werden diejenigen Systemelemente modelliert, die die spezifizierten Funktionen realisieren.

Die Analyse dieser beiden Spezifikationstechniken hat gezeigt, dass sie in enger Wechselwirkung zueinander stehen. Dies bedeutet, dass Änderungen an Modellen der einen Spezifikationstechnik oft auch Änderungen an Modellen der anderen Spezifikationstechnik nach sich ziehen. Zum Beispiel könnte ein Systemelement in der Systemstruktur vorhanden sein, obwohl es gar keine Funktion in der Funktionshierarchie gibt, die es erfüllt. Unglücklicherweise gibt es bisher jedoch keinen Formalismus, der sicherstellt, dass diese Abhängigkeiten automatisch berücksichtigt werden. Dies geschieht derzeit ausschließlich durch den Entwickler.

Darüber hinaus gibt es keine formale Unterstützung bei der Bestimmung der Systemelemente, die zur Realisierung der Funktionen verwendet werden sollen. Bei deren Auswahl muss sich auf das Wissen und die Erfahrung des Entwicklers verlassen werden, was sich aufgrund der Interdisziplinarität oft als schwierig erweist.

Die Lösung dieser Probleme ist die Aufgabe dieser Arbeit. Hierzu wurden insgesamt drei Schwerpunkte festgelegt. Der erste Schwerpunkt liegt in der Formalisierung der Funktionshierarchie und der Systemstruktur. Der zweite Schwerpunkt liegt in der Entwicklung eines Algorithmus zur automatischen Suche nach Systemelementen. Der dritte Schwerpunkt liegt in der Definition und Sicherstellung von Konsistenzbedingungen zwischen Modellen der beiden Spezifikationstechniken.

Danksagung

Mein Dank geht an Prof. Dr. Wilhelm Schäfer für die wissenschaftliche Betreuung in den vergangenen Jahren und für die Hilfe bei der Erstellung dieser Dissertation. Prof. Dr.-Ing. Jürgen Gausemeier danke ich für die Übernahme des Koreferats und für die anregenden Gespräche im Rahmen von Veranstaltungen des Sonderforschungsbereiches 614. Darüber hinaus danke ich unserer Sekretärin, Frau Jutta Haupt, für ihre tatkräftige Unterstützung in allen organisatorischen Fragen und ihren positiven Einfluss auf das gute Arbeitsklima in der Arbeitsgruppe.

Mein Dank geht an die (ehemaligen) Mitarbeiter unseres Fachgebiets Björn Axenath, Sven Burmester, Dr. Holger Giese, Stefan Henkler, Martin Hirsch, Dr. Ekkart Kindler, Florian Klein, Matthias Meyer, Ulrich Nickel, Dr. Jörg Niere, Vladimir Rubin, Daniela Schilling, Matthias Tichy, Dr. Jörg Wadsack, Robert Wagner, Lothar Wendehals und Prof. Dr. Albert Zündorf für viele interessante, avantgardistische Ideen in unseren zahlreichen Diskussionen und das Korrekturlesen dieser Arbeit. Darüber hinaus werden mir diverse Fußballspiele und Feierabendbiere in guter Erinnerung bleiben. Ferner danke ich allen Studien- und Diplomarbeitern sowie allen studentischen Hilfskräften für ihre Mitarbeit.

Weiterhin danke ich allen Mitarbeitern des Sonderforschungsbereiches 614, insbesondere Ursula Frank, Eckehard Münch, Alexander Redenius, Alexander Schmidt, Daniel Steffen und Henner Vöcking.

Mein ganz besonderer Dank gilt meinen Eltern, die mich immer unterstützt haben, auch wenn es mal gerade nicht so gut lief. Sie haben mich immer wieder aufgebaut und Kraft gegeben, insbesondere während der Fertigstellung dieser Dissertation.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

KAPITEL 1: EINLEITUNG

Was wir wissen ist ein Tropfen, was wir nicht wissen ein Ozean.

Isaac Newton

Die Entwicklung technischer Systeme ist heute gekennzeichnet durch das enge Zusammenwirken klassischer Ingenieursdisziplinen und der Informatik. Die so entstehenden Systeme werden als mechatronische Systeme bezeichnet und vereinen Technologien des Maschinenbaus, der Elektrotechnik, der Regelungstechnik und der Softwaretechnik. „*Im Maschinenbau werden zunehmend mechanische Lösungsprinzipien durch Informationstechnik substituiert und die Informationstechnik erlaubt dabei neue Lösungsprinzipien, die Quantensprünge im Preis-/Leistungsverhältnis ermöglicht*“ [GG97, S. 187]. Das Anti-Blockier-System (ABS), der Airbag oder die automatische Abstandskontrolle (ACC) sind Beispiele mechatronischer Systeme.

Eine der größten Herausforderungen in der Entwicklung mechatronischer Systeme ist es, die Möglichkeiten der verschiedenen Disziplinen zu kennen und diese so miteinander zu kombinieren, dass das synergetische Potential nutzbar gemacht wird. Aufgrund rasanter Weiterentwicklungen der Technologien innerhalb der einzelnen Disziplinen, unterliegen mechatronische Systeme jedoch immer kürzer werdenden Produktlebenszyklen. Dies hat zur Folge, dass die Kombinationsmöglichkeiten ständig überprüft und angepasst werden müssen. Dieses ständig neue Kombinieren erfordert allerdings viel Fachwissen über alle Disziplinen hinweg und stellt derzeitige Entwicklerteams vor große Herausforderungen.

Die Herausforderungen und zugleich die Chancen mechatronischer Systeme bestehen darin, das Potenzial disziplinübergreifender Zusammenarbeit gleichberechtigt zu nutzen und im Sinne eines „Simultaneous Engineering“ zu einem Gesamtoptimum zu führen [Bru96]. Um dieses Potential voll nutzen zu können, ist es von entscheidender Bedeutung, die Technologien der verschiedenen Disziplinen zu kennen, zu vergleichen und miteinander kombinieren zu können. Die so entstehende Komplexität erfordert jedoch eine neue Vorgehensweise bei der Entwicklung mechatronischer Systeme. Heutige mechatronische Systeme werden größtenteils von einer einzelnen Disziplin, vornehmlich dem Maschinenbau, geplant und entworfen. Erst in späteren Phasen werden die anderen Disziplinen integriert. Ein solches Vorgehen birgt jedoch große Risiken und kann zu steigenden Kosten und längeren Entwicklungszeiten führen [Kal98, S. 3]. Es existieren zu viele Abhängigkeiten, die berücksichtigt werden müssen, wodurch ein erhöhter Kommunikations- und Kooperationsbedarf entsteht. Dies führt zu der Erkenntnis, dass ein disziplinübergreifender Entwicklungsprozess benötigt wird. Ein erster Ansatz zu einer solchen Integration wurde in Form der VDI-Richtlinie 2206 realisiert [VDI03].

Die VDI-Richtlinie 2206 basiert auf dem aus der Softwareentwicklung bekannten V-Modell und hat dieses gemäß den Anforderungen der Mechatronik angepasst. Es werden grundsätzlich drei Phasen unterschieden: der Systementwurf, der domänenspezifische Entwurf und die Systemintegration. Insbesondere der Systementwurf ist dabei von entscheidender Bedeutung, denn hier werden die grundlegenden, disziplinübergreifenden Entscheidungen über das Lösungskonzept getroffen. Erst wenn sich alle beteiligten Entwickler über das Lösungskonzept geeinigt haben, kann die Arbeit im domänenspezifischen Entwurf begonnen werden.

Um einen gemeinsamen Konsens zu finden, bedarf es allerdings einer allgemein verständlichen Gesamtsicht auf das mechatronische Produkt, die von allen beteiligten Entwicklern verstanden und bewertet werden kann. Dies ist teilweise sehr schwer, denn abhängig davon aus welcher Disziplin der Entwickler stammt werden unterschiedliche Spezifikationstechniken für gleiche Sachverhalte verwendet [Ehr03, S. 14].

Um diesem Problem zu begegnen, wurden im Rahmen des Sonderforschungsbereich 614 „Selbstoptimierende Systeme des Maschinenbaus“[1] für die Phase des Systementwurfs neue Spezifikationstechniken entwickelt bzw. bestehende Spezifikationstechniken angepasst [SFB04], [GFS05]. Sachverhalte, die mit Hilfe dieser Spezifikationstechniken spezifiziert werden, sind Anforderungen, Umwelt, Zielsysteme, Funktionen, Wirkstrukturen, räumliche Darstellungen oder Anwendungsszenarien.

Mit Hilfe dieser Spezifikationstechniken werden dann während eines Systementwurfs konkrete Modelle erstellt, die in ihrer Gesamtheit die Prinziplösung eines mechatronischen bzw. selbstoptimierenden Systems darstellen. Die Verwendung dieser unterschiedlichen Modelle birgt jedoch eine Reihe von Problemen. Eines der größten Probleme stellt die Konsistenz der einzelnen Modelle zueinander dar. Nur wenn die Modelle zueinander konsistent sind, ist eine homogene, fehlerfreie Spezifikation des gesamten mechatronischen Systems möglich. Um diese Konsistenz gewährleisten zu können, bedarf es einer adäquaten Formalisierung der Modelle, ihrer Beziehungen zueinander und einer dadurch möglichen Rechnerunterstützung.

Im Sonderforschungsbereich 614 wurden die Beziehungen der einzelnen Spezifikationstechniken auf eine erste, informelle Art beschrieben. Hierbei wurde festgestellt, dass es zwischen einigen Spezifikationstechniken eher wenige Beziehungen gibt, zwischen anderen hingegen sehr starke. Einige Spezifikationstechniken stehen fast orthogonal zueinander, andere wiederum bauen aufeinander auf. Zwei Spezifikationstechniken die aufeinander aufbauen sind die Funktionsbeschreibung und die Wirkstruktur. In der Funktionsbeschreibung werden die funktionalen Anforderungen an das System definiert und in der Wirkstruktur werden die zu ihrer Realisierung gewählten Wirkprinzipien bzw. Lösungselemente modelliert. Ursprünglich stammen diese Spezifikationstechniken aus dem Maschinenbau, werden aber bereits seit langem auch im Entwurf mechatronischer Systeme eingesetzt.

Gemäß VDI-Richtlinie 2206 ist der Einsatz der Funktionsmodellierung der erste, zentrale Schritt in der Phase des Systementwurfs [VDI03, S. 25]. Mit ihrer Hilfe wird das mechatronische System möglichst abstrakt beschrieben, wobei nur die reine Funktionsweise von Interesse ist. Von möglichen Lösungen wird weitestgehend abstrahiert (lösungsneutral) um eine Vorfixierung und damit eine Einschränkung des möglichen Lösungsraumes zu verhindern. Funktionen werden in Form einer Funktionshierarchie (Baumstruktur) modelliert. Eine Funktionshierarchie stellt somit die Zerlegung einer komplexen Gesamtfunktion in weniger komplexe, handhabbare Teilfunktionen dar [PB03, S. 231].

Nach der Zerlegung der Gesamtfunktion in Teilfunktionen, ist der nächste zentrale Schritt in der Phase des Systementwurfs, die Ermittlung von Lösungen [VDI03, S. 27]. Jetzt müssen Systemelemente (z.B.: Hydraulikzylinder, Mehrgrößenregler, Steuergerät, etc.) ermittelt werden, die in der Lage sind die spezifizierten Funktionen zu realisieren. So könnte beispielsweise ein *Hydraulikzylinder* ausgewählt werden, um die Funktion *Kraft erzeugen* zu realisieren.

[1] Bei selbstoptimierenden Systemen handelt es sich um mechatronische Systeme mit inhärenter Teilintelligenz.

Abgesehen von der reinen Ermittlung und Auswahl der Systemelemente müssen diese nun noch zu einem Gesamtsystem verknüpft werden. Diese Verknüpfung erfolgt innerhalb der Wirkstruktur auf Basis unterschiedlicher Flussarten. Mit Hilfe von Informations-, Stoff- und Energieflüssen werden die Systemelemente miteinander verknüpft und geben somit einen ersten Überblick über die Struktur des Systems. Die Wirkstruktur kann als Konkretisierung der Funktionshierarchie verstanden werden.

Zur Ermittlung der Wirkstruktur existieren verschiedene Verfahren. Pahl und Beitz weisen auf die besondere Bedeutung der Literaturrecherchen, Methoden zur Analyse natürlicher und bekannter Systeme, intuitiv betonter Methoden und Katalogsuchen hin [PB03]. Mei Huang stellt einen ersten Ansatz zur automatischen Ermittlung von Wirkstrukturen vor, basierend auf einer erweiterten Funktionsstruktur (kanonische Funktionsstruktur) [Hu01]. Diesen und weiteren Verfahren ist gemein, dass sie die Expertise der Entwickler benötigen, um unterschiedlich geeignete Elemente zu ermitteln und aus dieser Menge von bekannten und bewährten Elementen diejenigen auszuwählen, die für die Funktionserfüllung verwendet werden können. Dies ist jedoch bei mechatronischen Systemen, wo die Lösung aus Elementen unterschiedlicher Disziplinen stammen können, nicht trivial. Aufgrund der Vielzahl von Kombinationsmöglichkeiten entsteht ein fast unüberschaubarer Lösungsraum. Nicht viele Entwickler, unabhängig aus welcher Disziplin sie ursprünglich stammen, sind in der Lage alle Vor- und Nachteile, Wechselwirkungen und Abhängigkeiten zu berücksichtigen und korrekt zu beurteilen [Ehr03].

Bei der Modellierung der Wirkstruktur muss berücksichtigt werden, dass die Verknüpfung der Elemente auf verschiedene Art und Weise erfolgen kann. Zum einen werden Elemente der Wirkstruktur durch physikalische Flüsse (Energie- und Stoffflüsse) und zum anderen durch Informationsflüsse miteinander verknüpft. Hierbei ist darauf zu achten, dass nicht jedes Element mit jedem beliebigen anderen Element verbunden werden darf. Beispielsweise darf ein Softwareelement (z.B. Regler) nicht mit Hilfe eines Stoffflusses mit einem Hardwareelement (z.B. Hydraulikzylinder) verbunden werden. Diese Einschränkungen müssen bei der Wirkstrukturmodellierung berücksichtigt werden, daher ist an dieser Stelle eine Formalisierung der Wirkstruktur sinnvoll.

Zusätzlich tritt das Problem auf, dass viele Elemente nur dann funktionieren, wenn gleichzeitig noch andere Elemente zur Verfügung stehen. So benötigt ein Hydraulikzylinder z.B. eine Pumpe, mit der ein bestimmter Öldruck erzeugt werden kann, oder ein Regler benötigt ein Steuergerät, auf dem er ausgeführt wird. Diese Abhängigkeiten müssen entsprechend verwaltet werden, um aufwändige Nachbesserungen zu einem späteren Zeitpunkt zu verhindern.

Schließlich ist zu bedenken, dass die Modellierung der Funktionshierarchie und der Wirkstruktur nicht streng sequentiell verläuft. Vielmehr existiert ein ständiger Wechsel zwischen der Funktionshierarchie und der Wirkstruktur. So kommen beispielsweise neue Funktionen hinzu, die die Auswahl anderer Systemelemente bedingt. Um bei diesen ständigen Änderungen den Überblick nicht zu verlieren, ist es erforderlich, die beiden Modelle konsistent zueinander zu halten. Es muss festgelegt und überprüft werden, welche Abhängigkeiten zwischen den Modellen existieren und wie mit diesen umzugehen ist. Dies setzt eine entsprechende Formalisierung der Spezifikationstechniken und ihrer Beziehungen voraus.

Ein erster Ansatz zur Verbesserung der Modellierung von Funktionshierarchien und Wirkstrukturen stellt Kallmeyer in seiner Arbeit vor, in der er u.a. eine Notation zur Beschreibung der Wirkstruktur entwickelt hat [Kal97]. Zusätzlich zu dieser Wirkstruktur, die

er selbst als Systemstruktur bezeichnet[2], hat er eine Vorgehensweise entwickelt, die den Übergang von einer Funktionshierarchie zur Systemstruktur beschreibt. Sowohl die dabei verwendete Funktionshierarchie, als auch die Systemstruktur und deren Abhängigkeiten sind umgangssprachlich bzw. informal beschrieben.

Durch die Erweiterung der Arbeit von Kallmeyer und einer entsprechenden Formalisierung der Funktionshierarchie, der Systemstruktur inkl. ihrer Zusammenhänge lassen sich die oben beschriebenen Probleme weitestgehend lösen bzw. in ihrer Komplexität reduzieren. Zusammengefasst sind dies:

- Durch die Kombination von Systemelementen aus den verschiedenen Disziplinen entsteht ein fast unüberschaubarer Lösungsraum. Durch die Formalisierung und der darauf basierenden Suche nach Systemelementen kann dieser Lösungsraum gehandhabt werden. Der Entwickler wird auf mögliche Lösungen hingewiesen, an die er sonst evtl. nicht gedacht hätte.
- Die durch die Suche ermittelten Systemelemente fördern die Kreativität des Entwicklers. Selbst wenn ein ermitteltes Systemelement nicht die optimale Lösung darstellt, so ist es dem Entwickler evtl. möglich auf dessen Basis eine neue Lösung zu kreieren.
- Viele Systemelemente können nur dann verwendet werden, wenn bestimmte, andere Systemelemente ebenfalls vorhanden sind. Diese Abhängigkeiten machen den Systementwurf äußerst komplex, sobald eine entsprechend große Menge an Systemelementen zum Einsatz kommt. Der Entwickler muss wissen, welche Abhängigkeiten existieren. Werden Abhängigkeiten übersehen, kann dies erhebliche Kosten nach sich ziehen. Durch die Formalisierung lässt sich dieses Problem lösen, da in der Systemelementbibliothek die Abhängigkeiten hinterlegt und der Entwickler entsprechend darauf aufmerksam gemacht werden kann.
- Durch das Zuweisen von Systemelementen zu Funktionen, unter Berücksichtigung der Abhängigkeiten, können sehr umfangreiche Funktionshierarchien und Systemstrukturen entstehen. Aufgrund von Anforderungsänderungen kann es jedoch vorkommen, dass Funktionen gelöscht und dadurch die zugewiesenen Systemelemente überflüssig werden. Durch die Formalisierung und der Definition von Konsistenzbedingungen lässt sich sicherstellen, dass keine Systemelemente in der Systemstruktur verbleiben, die aufgrund gelöschter Funktionen nicht mehr benötigt werden.
- Durch die Zuweisung von Systemelementen zu Funktionen entsteht Stück für Stück das mechatronische System. Allerdings lässt sich nicht immer mühelos nachvollziehen, welche Funktion bereits durch Systemelemente erfüllt wird und welche nicht. Dies ist durch die Formalisierung möglich. Mit ihrer Hilfe lässt sich jederzeit feststellen, wie der aktuelle Entwicklungsstand ist.

Zusammenfassend lässt sich feststellen, dass durch eine Formalisierung der Funktionshierarchie, der Systemstruktur und deren Beziehungen die Modellierung während der Phase des Systementwurfs deutlich verbessert wird. Durch eine Formalisierung wird eine Werkzeugunterstützung möglich, die den Entwickler bei seiner Arbeit unterstützt. So ist beispielsweise eine automatische Plausibilitätsprüfungen möglich, wodurch verhindert wird, dass inkompatible Elemente in der Systemstruktur miteinander verknüpft werden. Des Weiteren lässt sich die Ermittlung von Elementen zur Realisierung von Funktionen automatisieren. Nur durch eine entsprechende Formalisierung ist ein Werkzeug hierzu in der

[2] Im Folgenden wird in dieser Arbeit ebenfalls das Wort Systemstruktur verwendet.

Lage. Schließlich wird es erst durch eine Formalisierung möglich die Konsistenz zwischen der Funktionshierarchie und der Systemstruktur zu gewährleisten. Nur wenn klar definiert ist, was ein konsistenter Zustand ist, lässt sich dies auch automatisch, mit Hilfe eines entsprechenden Werkzeugs, sicherstellen.

1.1 Zielsetzung

Ziel dieser Arbeit ist die Formalisierung der Spezifikationstechniken zur Modellierung der Funktionshierarchie und der Systemstruktur. Darüber hinaus soll ein Mechanismus entwickelt werden, der auf Basis einer Funktion automatisch diejenigen Systemelemente ermittelt, die in der Lage sind, diese Funktion zu realisieren. Dabei sollen auch mögliche Abhängigkeiten zwischen den Systemelementen berücksichtigt werden. Schließlich sollen die Beziehungen zwischen der Funktionshierarchie und der Systemstruktur so formalisiert werden, dass die Konsistenz zwischen ihnen sichergestellt werden kann.

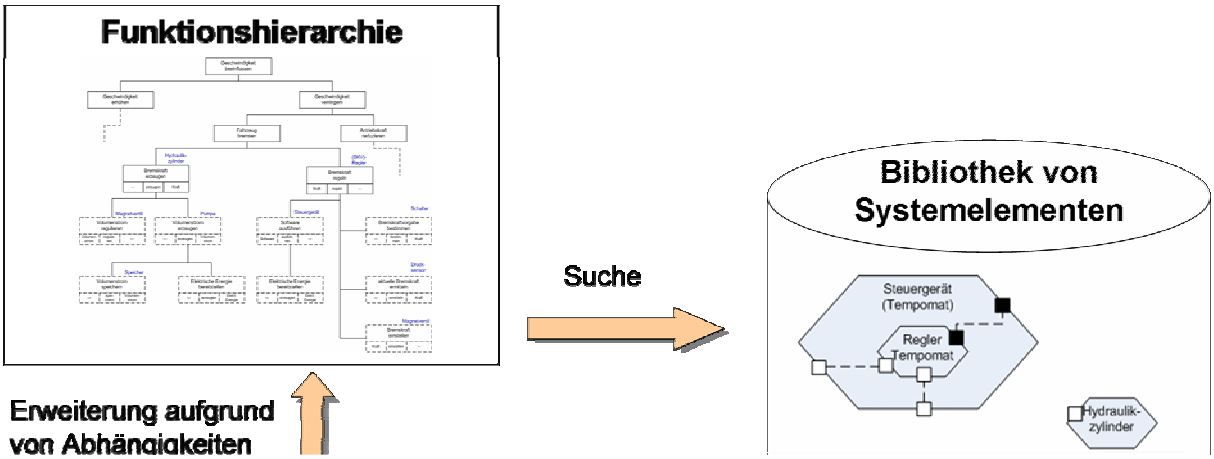
Die konzeptionelle Grundlage zur Beschreibung der Funktionshierarchie stammt aus den VDI-Richtlinien 2222 Blatt-1 [VDI97] und der VDI-Richtlinie 2206 [VDI04]. Zur Beschreibung der Systemstruktur bildet die Arbeit von Kallmeyer [Kal98] die konzeptionelle Basis. Zur Formalisierung beider Spezifikationstechniken werden Modelle der Unified Modeling Language [UML03] verwendet, wobei insbesondere Klassendiagramme zum Einsatz kommen.

Den Ausgangspunkt für die Suche nach Systemelementen stellt die Funktionshierarchie dar. Bei dieser Suche wird für eine gegebene Funktion ermittelt, welche Systemelemente diese Funktion realisieren könnten. Zu diesem Zweck wird eine Bibliothek von Systemelementen definiert, in der die Systemelemente abgelegt werden. Die Suche erfolgt mit Hilfe eines graphbasierten Algorithmus, der die Bibliothek nach passenden Elementen durchsucht. Das Suchergebnis wird dann dem Entwickler zur Auswahl präsentiert. Auf diese Weise entsteht sukzessiv die Systemstruktur, indem für jede Funktion ein Systemelement ermittelt und ausgewählt wird. Formal wird der Suchalgorithmus mit Hilfe von Graphtransformationsregeln beschrieben [Roz97], [Zün02].

Systemelemente, die ausgewählt und in die Systemstruktur integriert werden, können von anderen Systemelementen abhängig sein. Diese Abhängigkeiten werden mit Hilfe von Hilfsfunktionen ausgedrückt und sind ebenfalls in der Bibliothek hinterlegt. Treten Hilfsfunktionen auf (z.B. die Funktion „Elektrische Energie bereitstellen“ im Falle einer Pumpe), werden diese in die Funktionshierarchie integriert, was bedeutet, dass für diese Hilfsfunktionen wiederum Systemelemente ermittelt und ausgewählt werden müssen.

Ein weiterer Schwerpunkt der Arbeit liegt in der Sicherstellung der Konsistenz zwischen Modellen der Funktionshierarchie und der Systemstruktur. Zu diesem Zweck werden Konsistenzbedingungen definiert und diese mit Hilfe von Graphtransformationsregeln formalisiert. Somit können die Funktionshierarchie oder die Systemstruktur geändert werden, ohne dass der Überblick darüber verloren geht, welche Funktionen bereits durch Systemelemente realisiert werden und welche noch nicht.

Zusammenfassend beschäftigt sich diese Arbeit mit dem Systementwurf mechatronischer Systeme. Besonderes Augenmerk wird dabei auf die Formalisierung der Funktionshierarchie und Systemstruktur inkl. ihrer Beziehungen gelegt. Die Vorgehensweise bei der Modellierung der beiden Modelle gleicht einem Kreislauf (vgl. Abbildung 1). Begonnen wird mit der Modellierung der Funktionshierarchie. Darauf aufbauend werden Systemelemente für die einzelnen Funktionen ermittelt. Durch die Wahl dieser Systemelemente wird aufgrund von Abhängigkeiten die Funktionshierarchie erweitert bzw. geändert. Für die hinzugekommenen Funktionen müssen nun wiederum Systemelemente ermittelt werden.


Funktionshierarchie
Suche
Bibliothek von Systemelementen
Steuergerät (Tempomat)
Regler Tempomat
Hydraulik-zylinder
Erweiterung aufgrund von Abhängigkeiten

In Kapitel 7 wird ein Suchalgorithmus formal beschrieben, der die Suche nach Systemelementen auf Basis von Funktionen ermöglicht. Die Formalisierung dieses Suchalgorithmus beruht auf Graphtransformationsregeln.

In Kapitel 8 wird die Konsistenz zwischen den beiden Spezifikationstechniken behandelt. Zuerst werden verschiedene Konsistenzbedingungen aufgestellt und diese dann mit Hilfe von Graphtransformationsregeln formalisiert.

In Kapitel 9 wird die im Rahmen dieser Arbeit entstandene Werkzeugunterstützung vorgestellt. Das entstandene Werkzeug basiert auf der Entwicklungsumgebung Eclipse und wurde in Form zweier Plug-Ins realisiert.

Kapitel 10 schließt die Arbeit mit einer Zusammenfassung der Ergebnisse und offener Probleme sowie Verbesserungen und Perspektiven des Ansatzes ab.

KAPITEL 2: MECHATRONISCHE SYSTEME

Das folgende Kapitel soll grundlegendes Verständnis darüber erzeugen, wie mechatronische Systeme prinzipiell entworfen werden. Hierzu wird zu Beginn kurz erläutert was mechatronische Systeme sind und welche Struktur sie besitzen. Anschließend wird erläutert wie mechatronische Systeme systematisch entworfen werden, d.h. welche Vorgehensweisen diesbezüglich bisher existieren. Am Ende des Kapitels werden die konzeptionellen Schwächen der Vorgehensweisen zusammenfassend dargestellt.

Einige der hier vorgestellten Ansätze stellen für diese Arbeit nicht nur den Stand der Technik, sondern auch Grundlagen dar auf denen aufgebaut wird. Aus diesem Grund sind sie teilweise sehr ausführlich erläutert.

2.1 Mechatronik

Das Wort Mechatronik ist ein 1969 in Japan entstandenes Kunstwort. Der Japaner Ko Kikuchi, Präsident der YASKAWA Electric Corporation, prägte diesen Begriff indem er „Mechanik“ und „Elektronik“ zu einem Wort zusammenfasste [HTF96]. Mit dem Aufkommen der Mikroelektronik und besonders der Mikroprozessortechnik ist die Informationstechnik als weiterer Bestandteil der Mechatronik hinzugekommen.

Ziel der Mechatronik ist die Entwicklung von Systemen die in der Lage sind die Potentiale aller Disziplinen zu nutzen um leistungsfähigere Systeme konstruieren zu können. Die Mechatronik nutzt somit die Synergien aus dem Zusammenwirken der klassischen Ingenieurwissenschaften Maschinenbau, Elektrotechnik und Informationstechnik [VDI03].

2.1.1 Definition Mechatronik

Zurzeit besteht noch kein allgemeiner Konsens im Bezug auf eine einheitliche Definition des Begriffes Mechatronik. Es existieren viele ähnliche aber in Details voneinander abweichende Definitionen [VDI03], die häufig Gegenstand lebhafter Diskussionen sind [TH96].

Harashima, Tomizuka und Fukuda definierten 1996 [HTF96]:

„[Mechatronics is]...the synergetic integration of mechanical engineering with electronic and intelligent computer control in the design and manufacturing of industrial products and processes.“

Im deutschsprachigen Raum ist folgende Definition von Isermann aus dem Jahre 1999 zu finden [Ise99]:

„Mechatronik ist ein interdisziplinäres Gebiet, bei dem folgende Disziplinen zusammenwirken: Mechanische und mit ihnen gekoppelte Systeme, Elektronische Systeme, Informationstechnik. Dabei ist das mechanische System im Hinblick auf die Funktionen dominierend. Es werden synergetische Effekte angestrebt, die mehr beinhalten als die reine Addition der Disziplinen“.

Diese beiden Definitionen stehen stellvertretend für eine Vielzahl von Definitionen. Abgesehen von kleineren Nuancen liegt der Schwerpunkt bei allen Definitionen aber auf der Zusammenarbeit der Disziplinen Maschinenbau, Elektrotechnik und Informationstechnik (vgl. Abbildung 2). Durch ”Synergien“ dieser Disziplinen soll die interdisziplinäre Herangehensweise aktiviert werden, womit die funktionellen Möglichkeiten gemeint sind, die in den einzelnen Disziplinen allein kaum realisierbar sind.

Zu beachten ist allerdings, dass diese drei Disziplinen in eine Vielzahl weiterer Teildisziplinen unterteilt werden können. So kann der Maschinenbau in die Teildisziplinen Mechanik, Feinwerktechnik oder Fluidtechnik unterteilt werden. Die Elektrotechnik lässt sich in Mikroelektronik und Leistungselektronik unterteilen. Zur Disziplin Informationstechnik gehören große Gebiete der Informatik, wie z.B. die Softwaretechnik, Programmiersprachen und Compilerbau, Betriebssysteme oder Kryptologie, so dass es aus heutiger Sicht nahe liegender wäre, Informationstechnik durch Informatik zu ersetzen.

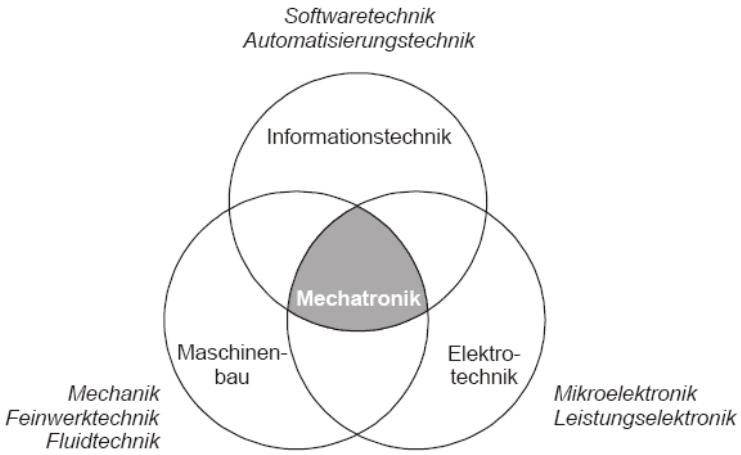


Abbildung 2: Synergie aus dem Zusammenwirken verschiedener Disziplinen [Ise99]

2.1.2 Grundstruktur

Die Grundstruktur mechatronischer Systeme besteht aus einem Grundsystem, Sensoren, Aktoren und einer Informationsverarbeitung. Diese Grundstruktur wird auch als „Mechatronisches Funktionsmodul“ (MFM) bezeichnet [LHL01] und ist in Abbildung 3 skizziert.

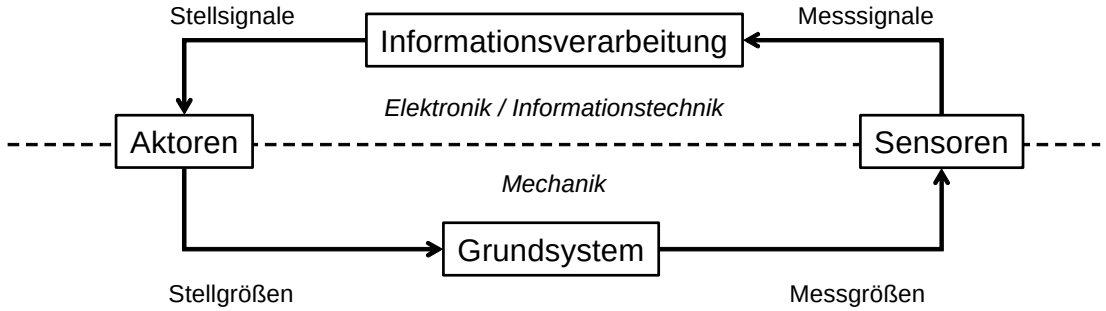


Abbildung 3: Grundstruktur eines mechatronischen Systems (MFM)

Das *Grundsystem*, meist mechanisch[3], übt die Hauptfunktion des Systems aus. Es wird mit Hilfe von Sensoren überwacht und mit Hilfe von Aktoren beeinflusst.

Die *Sensoren* nehmen Messgrößen über das Grundsystem auf und übertragen diese an die Informationsverarbeitung. Sensoren können dabei sowohl physisch vorhandene Messwertaufnehmer als auch reine Softwaresensoren (sog. Beobachter, siehe z.B. [Ise88]) sein.

[3] Das Grundsystem kann neben einem mechanischen System aber auch ein pneumatisches, hydraulisches oder sonstiges physikalisches System sein. Im Sinne des komponentenorientierten Entwurfs ermöglicht dies auch die Benutzung eines weiteren mechatronischen Systems als Grundsystem [VDI03].

Mit Hilfe der Sensordaten (Messsignale) errechnet die *Informationsverarbeitung* anschließend Werte zur Optimierung des Verhaltens des Grundsystems. Die Informationsverarbeitung kann im einfachsten Fall aus einer simplen elektronischen Schaltung bestehen, aber auch aus einem eingebetteten System, das mit spezieller Software die Sensordaten verarbeitet.

Die von der Informationsverarbeitung berechneten Werte (Stellsignale) werden an die *Aktoren* weitergegeben. Diese sind dann in der Lage direkt auf das Grundsystem einzuwirken (Stellgrößen), um die Optimierungen umzusetzen.

2.1.3 Hierarchische Struktur mechatronischer Systeme

Mechatronische Systeme setzen sich im Allgemeinen nicht nur aus einer Grundstruktur bzw. MFM zusammen, sondern aus mehreren (z.B.: ABS und ESC im Fahrzeug). Diese MFM's nutzen dabei Daten (z.B.: Messsignale, Stellsignale) anderer MFM's um dadurch Redundanz und höhere Kosten zu vermeiden. Um die dadurch entstehende Komplexität beherrschen zu können schlägt Lückel [LKS00], [LK03] eine hierarchische Struktur vor (vgl. Abbildung 4).

Die unterste Ebene dieser Struktur bilden die mechatronischen Funktionsmodule (MFM), die bereits in 2.1.2 ausführlich erklärt wurden.

Durch die informationstechnische und/oder mechanische Vernetzung der MFM gelangt man zur nächst höheren hierarchischen Ebene. Diese wird als autonomes mechatronisches System (AMS) bezeichnet.

Die oberste Ebene bilden die vernetzten mechatronischen Systeme (VMS) die durch rein informationstechnisch gekoppelte autonome mechatronischen Systeme entstehen.

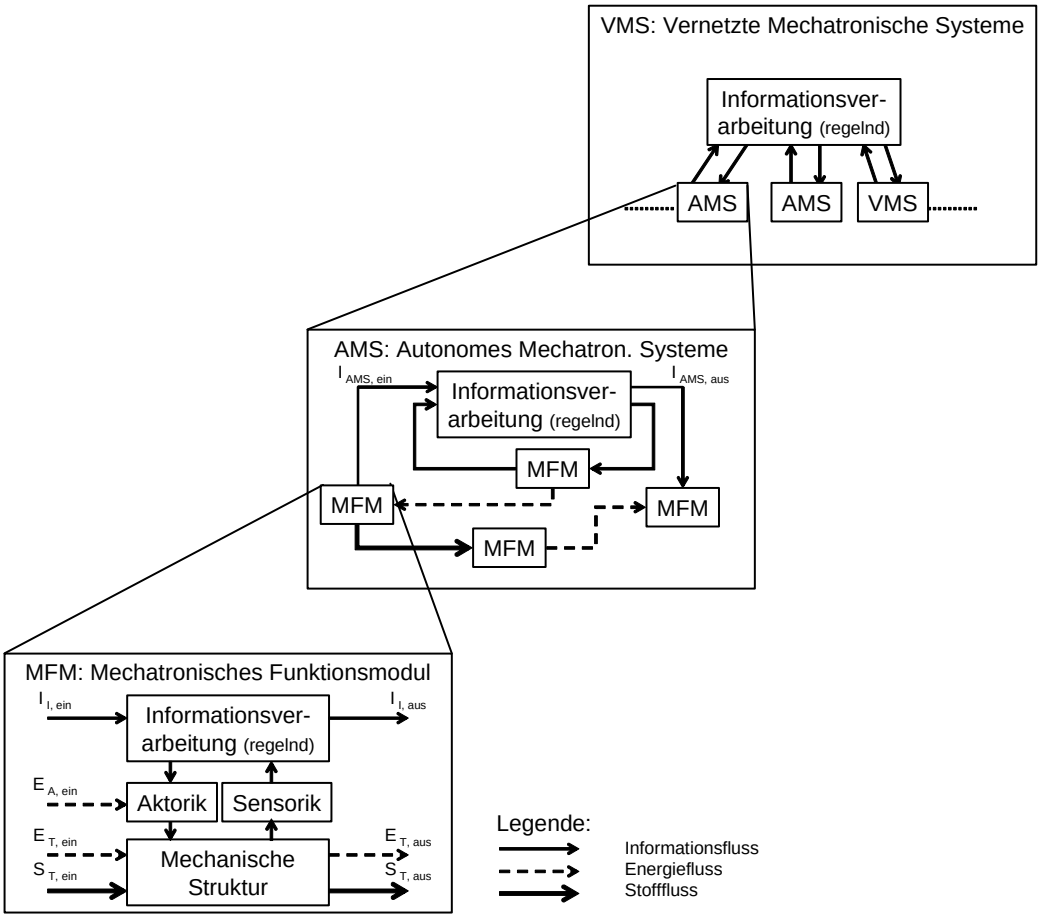


Abbildung 4: Hierarchische Struktur mechatronischer Systeme (aus [VDI04])

Mit Hilfe dieser Hierarchisierung ist es möglich komplexe mechatronische Systeme übersichtlich zu beschreiben. Man erhält eine Strukturierungsform die dabei hilft das Zusammenspiel der einzelnen Komponenten (MFMs, AMSs, VMSs) zu modellieren.

Nachdem nun ausführlich die Struktur mechatronischer Systeme dargestellt wurde, schließt sich die Frage an, wie solche Systeme eigentlich entwickelt werden. Wann wird z.B. entschieden, wie das Grundsystem aufgebaut ist und mit welchen Sensoren bzw. Aktoren darauf Einfluss genommen werden soll. Die Antworten hierzu behandelt der folgende Abschnitt.

2.2 Entwicklungsprozess

Aufgrund des vernetzten Zusammenspiels unterschiedlicher Disziplinen ist die Entwicklung mechatronischer Systeme sehr komplex. Diese Komplexität ergibt sich aufgrund der im Vergleich zu mechanischen Systemen größeren Anzahl von verkoppelten Elementen, die in unterschiedlichen Disziplinen realisiert werden. Aufgrund dieser Tatsache besteht Bedarf nach einem disziplinübergreifenden Entwicklungsprozess, der die Aufgaben, Probleme und Anforderungen aller Disziplinen berücksichtigt [VDI04].

Im Laufe der Jahre haben sich für jede Disziplin eigene Entwicklungsprozesse entwickelt. Diese Entwicklungsprozesse sind genau auf die Anforderungen und Eigenschaften der jeweiligen Disziplin abgestimmt. Dies hat sich in der Vergangenheit zu einer disziplinspezifischen Denkweise geführt. Ausgehend von dieser Denkweise wurde versucht einen disziplinübergreifenden Prozess zur Entwicklung mechatronische Systeme zu definieren. Das führte dazu, dass dieser Entwicklungsprozess von einer einzelnen Disziplin dominiert wurde. Die Entwickler dieser Disziplin legten das Maßgebliche Vorgehen fest, ohne dabei die Belange der anderen Disziplinen zu berücksichtigen. Ehrlenspiel [Ehr95] bezeichnet dies als „Mauerndenken“ (vgl. Abbildung 5). Das Resultat sind unvollständige oder fehlerhafte Produkte die aufgrund langwieriger Iterationen Zeit-, Qualitäts- und Kostenprobleme verursachen.

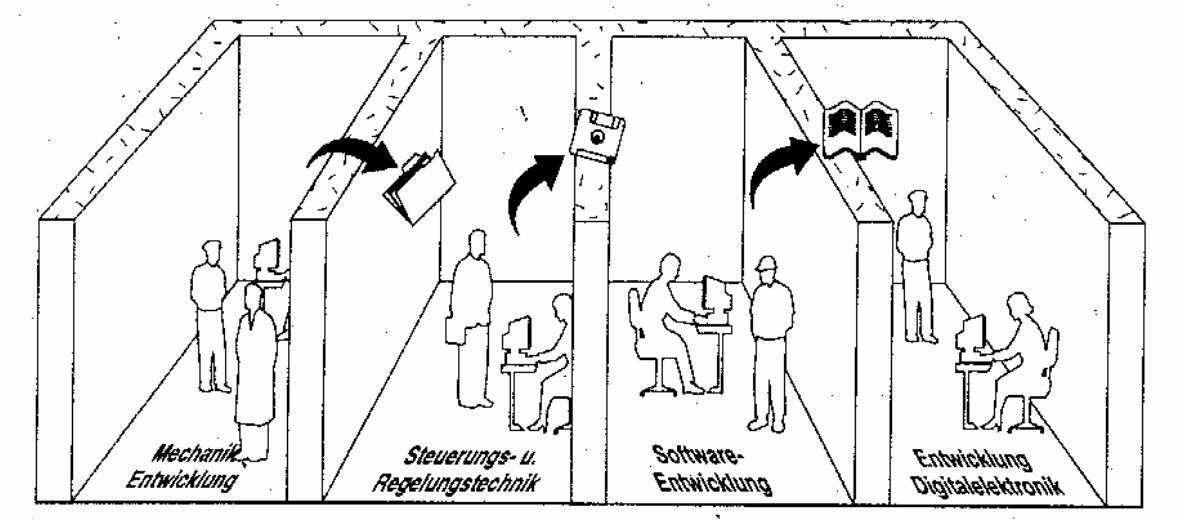


Abbildung 5: Mauerndenken nach Ehrlenspiel

Um diesem Problem zu begegnen wird ein Entwicklungsprozess gesucht der alle Disziplinen gleichermaßen berücksichtigt.

In den folgenden Abschnitten werden erst die disziplinspezifischen und dann die disziplinübergreifenden Entwicklungsprozesse vorgestellt, wobei der Fokus auf den disziplinübergreifenden liegt. Diese werden ausführlicher dargestellt und bilden den methodischen Rahmen in den sich diese Arbeit einbettet.

2.2.1 Disziplinspezifische Entwicklungsprozesse

Die Grundlage disziplinübergreifender Vorgehensweisen bilden die in den einzelnen Disziplinen entwickelten und etablierten Entwicklungsprozesse.

Die erste der drei beteiligten Disziplinen ist der Maschinenbau. Die Aufgabe des Maschinenbaus ist der Entwurf und die Herstellung von Maschinen aller Art. Er setzt dabei physikalische Gesetzmäßigkeiten, insbesondere aus den Teilgebieten Mechanik und Thermodynamik, für die Konstruktion und die Simulation technischer Anlagen ein. Um diese Maschinen und Anlagen systematisch erstellen zu können wurden über Jahre hinweg eine Reihe von Vorgehensweisen entwickelt, wobei sich viele dieser Vorgehensweise stark ähneln und nur in Details voneinander abweichen. Exemplarisch seien hier die Vorgehensmodelle von Pahl/Beitz [PB03], Roth [Rot00], Koller [Kol98] oder Ehrlenspiel [Ehr03] genannt. Die VDI-Richtlinie 2221 [VDI93] bzw. 2222 Blat-1 [VDI97] greifen einige dieser Ansätze auf und fassen sie in einer einheitlichen Richtlinie zusammen[4].

Die zweite Disziplin ist die Elektrotechnik. Sie beschäftigt sich mit der technischen Anwendung der physikalischen Grundlagen und Erkenntnisse der Elektrizitätslehre, das heißt sie nutzt die Erscheinungsformen und Wirkungen elektrischer Ladungen und Ströme, die von ihnen erzeugten elektrischen und magnetischen Felder sowie ihre wechselseitigen elektromagnetischen Beeinflussungen. Die Elektrotechnik hat insbesondere mit der Entwicklung der Mikroelektronik besondere Bedeutung erfahren. Mikroelektronische Systeme befinden sich heutzutage in vielen technischen Produkten (z.B. in Flugzeugen oder Kraftfahrzeugen). Die Mikroelektronik lässt sich prinzipiell in die analoge und die digitale Elektronik unterteilen, wobei letztere immer mehr an Bedeutung gewinnt. Zur systematischen Entwicklung analoger und digitaler, elektronischer Systeme wurden eine Reihe von Vorgehensweisen definiert. Exemplarisch seien hier die Vorgehensweisen von Eschermann [Esc93], Chang [Cha97], Armstrong [Arm93] und Heinzl [Hei84] erwähnt.

Die dritte Disziplin ist die Informationstechnik. Die Informationstechnik besteht zum einen aus der Softwaretechnik und zum anderen aus der Automatisierungs- bzw. Regelungstechnik. Die Softwaretechnik ist das Gebiet der Informatik, das sich mit Methoden und Werkzeugen für das ingenieurmäßige Entwerfen, Herstellen und Implementieren von Software befasst. Die in diesem Bereich etablierten Vorgehensweisen sind das Wasserfallmodell [PB96], V-Modell [Ver00], Spiralmodell [Boe86], [Boe88] oder der objektorientierte Softwareentwurf [Boo96].

Die Automatisierungs- bzw. Regelungstechnik hat die Aufgabe, hauptsächlich in technischen Anlagen physikalische Größen (Regelgrößen) trotz des Einflusses äußerer Störungen (Störgrößen) konstant zu halten oder den zeitlichen Verlauf vorgegebener Größen (Führungsgrößen) möglichst genau nachzuführen. Die in diesem Bereich etablierten Vorgehensweisen sind in den Arbeiten von Föllinger [Föl94], Dörrscheidt [DL93] oder Samal [SB96] nachzulesen.

Detaillierte Angaben und Analysen der disziplinspezifischen Vorgehensweisen sind nicht Fokus dieser Arbeit. Aus diesem Grund sei hier auf die Literatur verwiesen, wie z.B. auf die Arbeiten von Kallmeyer [Kal98] oder Huang [Hua01].

Einordnung: Die grundlegende Erkenntnis der vorgestellten Vorgehensweisen ist, dass diese sehr speziell und detailliert auf die Anforderungen der einzelnen Disziplinen ausgerichtet sind und sich in den meisten Fällen nicht aufeinander abstimmen lassen. Eine solche Abstimmung ist aber zwingend notwendig, da es eine Reihe von Abhängigkeiten

[4] Roth kommt zu dem Schluss: *„Die Ablaufpläne von Roth und Pahl/Beitz stimmen mit dem Ablaufplan der Richtlinie VDI 2221 genau überein.“* [Rot00].
Huang schreibt: *„Durch die Richtlinienarbeit VDI 2221 und VDI 2222, in der die meisten Vertreter der einzelnen Schulen involviert waren, ist ein prinzipieller Konsens erzielt worden.“* [Hua01]

zwischen den in den verschiedenen Disziplinen entwickelten Elementen gibt. So kann z.B. die Entwicklung eines Steuergeräts (Disziplin: Elektrotechnik) entscheidenden Einfluss auf die Entwicklung der Software (Disziplin: Informationstechnik) haben, die später auf ihm ausgeführt werden soll. Wird beispielsweise ein Steuergerät mit wenig Speicher entwickelt, dann muss die Softwareentwicklung dies entsprechend berücksichtigen, da sonst die beiden Elemente nicht zusammenpassen.

Anhand dieser Abhängigkeiten lässt sich erkennen, dass die Prozesse zur Entwicklung mechatronischer Systeme aufeinander abgestimmt werden müssen.

2.2.2 Disziplinübergreifende Entwicklungsprozesse

Eine große Anzahl an disziplinübergreifenden Entwicklungsprozessen gibt es bisher nicht. Die bekanntesten und verbreiteten werden im Folgenden vorgestellt.

2.2.2.1 VDI / VDE-Richtlinie 2422

Die VDI / VDE - Richtlinie 2422 [VDI94] beschreibt die Entwicklung mechanischer Systeme, die durch Mikroelektronik gesteuert werden. Die Richtlinie stellt eine Vorgehensweise vor, um den Problemen der zunehmenden Verzahnung von Mechanik und Mikroelektronik Herr zu werden. Dieses Vorgehensmodell basiert auf der VDI-Richtlinie 2222 (aus dem Jahr 1977) und verfeinert diese, indem es Arbeitsschritte der Disziplinen Elektronik und Softwaretechnik hinzufügt.

Zentrales Element ist das Steuergerät. Ein Steuergerät verbindet die mechanischen Komponenten eines Systems mit der Software. Die aufgespielte Software ist in der Lage Sensorwerte zu analysieren und entsprechende Stellsignale zu berechnen, die dann die nötigen Reaktionen auslösen.

Abbildung 6 zeigt, welche Entwurfsschritte durchzuführen sind um ein Steuergerät zu entwickeln. Dabei wird in die drei Gebiete Softwareentwurf, Schaltungsentwurf und Elektromechanischer Entwurf unterteilt. Erst nach deren Ausarbeitung wird das Gerät zusammengesetzt und geprüft.

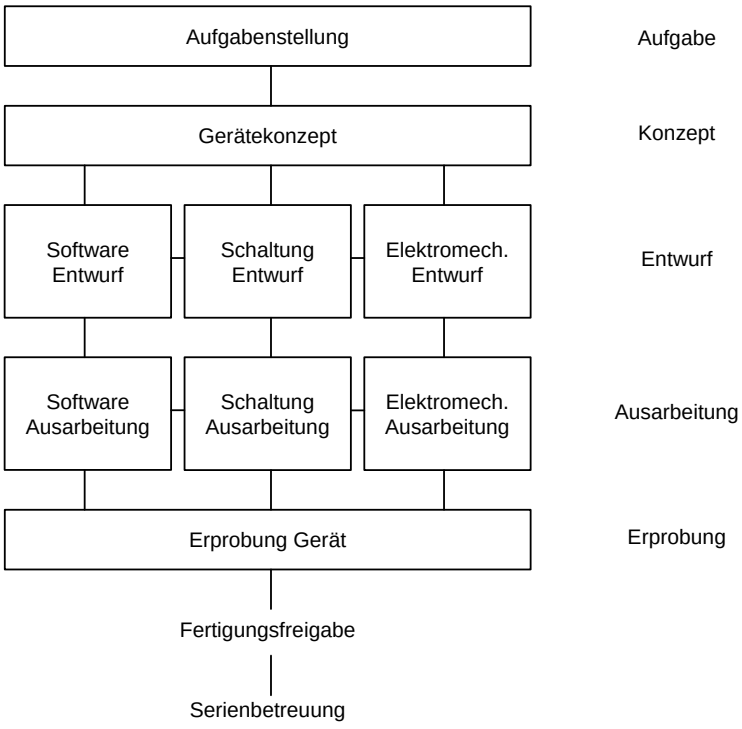


Abbildung 6: Entwicklung von Geräten mit Steuerung durch Mikroelektronik [VDI94]

Einordnung: Die VDI / VDE - Richtlinie 2422 integriert die Entwicklung von Mechanik und Mikroelektronik, wobei sie die disziplinspezifischen Vorgehensweisen beibehält und versucht diese aufeinander abzustimmen. Ein lösungsneutraler Entwurf mechatronischer Systeme ist hiermit jedoch nicht möglich, da sehr detaillierte Kenntnisse über das zu entwickelnde System bereits vorliegen müssen. Z.B. ist die Entwicklung von präzisen Schaltungen im Rahmen des Schaltungsentwurfs nur mit Hilfe konkreter Angaben möglich. Die Entwicklung von Funktionshierarchien, Funktionsstrukturen oder Wirkstrukturen wird nur oberflächlich angesprochen. Konkrete Angaben wie diese modelliert werden sollen, werden nicht gemacht.

2.2.2.2 VDI-Richtlinie 2206

Die VDI-Richtlinie 2206 [VDI04] stellt einen praxisorientierten Leitfaden für die systematische Entwicklung mechatronischer Produkte dar. Ziel dieser Richtlinie ist es das disziplinübergreifende Entwickeln mechatronischer Systeme methodisch zu unterstützen. Die Ausgangssituation die zur Erstellung dieser Richtlinie führte war die bisherige, mangelnde Integration der beteiligten Disziplinen in die Entwicklungssystematik Mechatronischer Systeme[5]. Die Richtlinie soll keine etablierten Richtlinien (wie z.B. VDI-Richtlinie 2221, VDI-Richtlinie 2222) oder disziplinspezifische Vorgehensweisen ersetzen, sondern hat vielmehr den Anspruch diese systematisch zusammenzuführen.

Das Hauptaugenmerk der Richtlinie liegt bei den Vorgehensweisen, Methoden und Werkzeugen der frühen Phase des Entwickelns, dem Systementwurf.

Die Richtlinie baut sich, neben Einführung und Beispielen, im Wesentlichen aus vier Teilen auf:

1. Vorgehensmodell: Hier werden ein Mikro- und ein Makrozyklus (siehe unten) vorgestellt, die die Vorgehensweisen der Entwicklung beschreiben. Der Mikrozyklus stellt eine Leitlinie für das Bearbeiten einzelner Teilaufgaben dar. Der Makrozyklus hingegen beschreibt den Gesamtablauf der Entwicklung eines mechatronischen Systems.
2. Modellbasierter Systementwurf: Dieser Teil gibt Ratschläge zur Modellierung von Teilaspekten des Systems. Diese Modelle sollen dazu dienen – vor allem im mechanischen und elektrotechnischen Bereich –, mit Hilfe von Analysen und Simulationen Voraussagen über das Verhalten des Systems machen zu können.
3. Werkzeuge: Programme, die im mechatronischen Entwurf hilfreich sind, werden in diesem Abschnitt klassifiziert und beschrieben. Der Abschnitt dient im Wesentlichen dazu, dem Entwickler mechatronischer Systeme einen Überblick über die verfügbaren Programme zu geben.
4. Organisation: Informationen über die Organisierung von Projektteams oder die Einbindung von Zulieferfirmen werden in diesem Teil beschrieben.

Im folgenden werden die für diese Arbeit relevanten Inhalte der Richtlinie detaillierter vorgestellt.

[5] Die VDI-Richtlinie 2206 ist die erste VDI-Richtlinie, deren zentrales Themengebiet die Mechatronik ist. Die VDI-Richtlinie 2422, ”Entwicklungsmethodik für Geräte mit Steuerung durch Mikroelektronik“, [VDI94] kann als Vorgänger der Richtlinie 2206 angesehen werden.

Der Makrozyklus

Eine Richtschnur für das grundsätzliche Vorgehen bietet das aus der Softwareentwicklung übernommene und an die Anforderungen der Mechatronik angepasste V-Modell (siehe Abbildung 7), das die logische Abfolge wesentlicher Teilschritte bei der Entwicklung mechatronischer Systeme beschreibt [VDI04, S. 20].

Das V-Modell setzt sich aus drei Phasen zusammen. Begonnen wird mit dem auf den Anforderungen basierenden *Systementwurf*. Ziel hierbei ist das Festlegen eines disziplinübergreifenden Lösungskonzeptes, welches die wesentlichen physikalischen und logischen Wirkungsweisen beschreibt. Daran schließt sich der *domänenspezifische Entwurf* an, in dem das entwickelte Lösungskonzept innerhalb der jeweils zuständigen Disziplinen weiter ausgearbeitet wird. In der *Systemintegration* werden die Ergebnisse aus den Disziplinen dann zu einem Gesamtsystem zusammengesetzt. Dieses Gesamtsystem wird daraufhin getestet und mit dem spezifizierten Lösungskonzept abgeglichen. Waren alle Tests erfolgreich, ist die dritte und letzte Phase beendet.

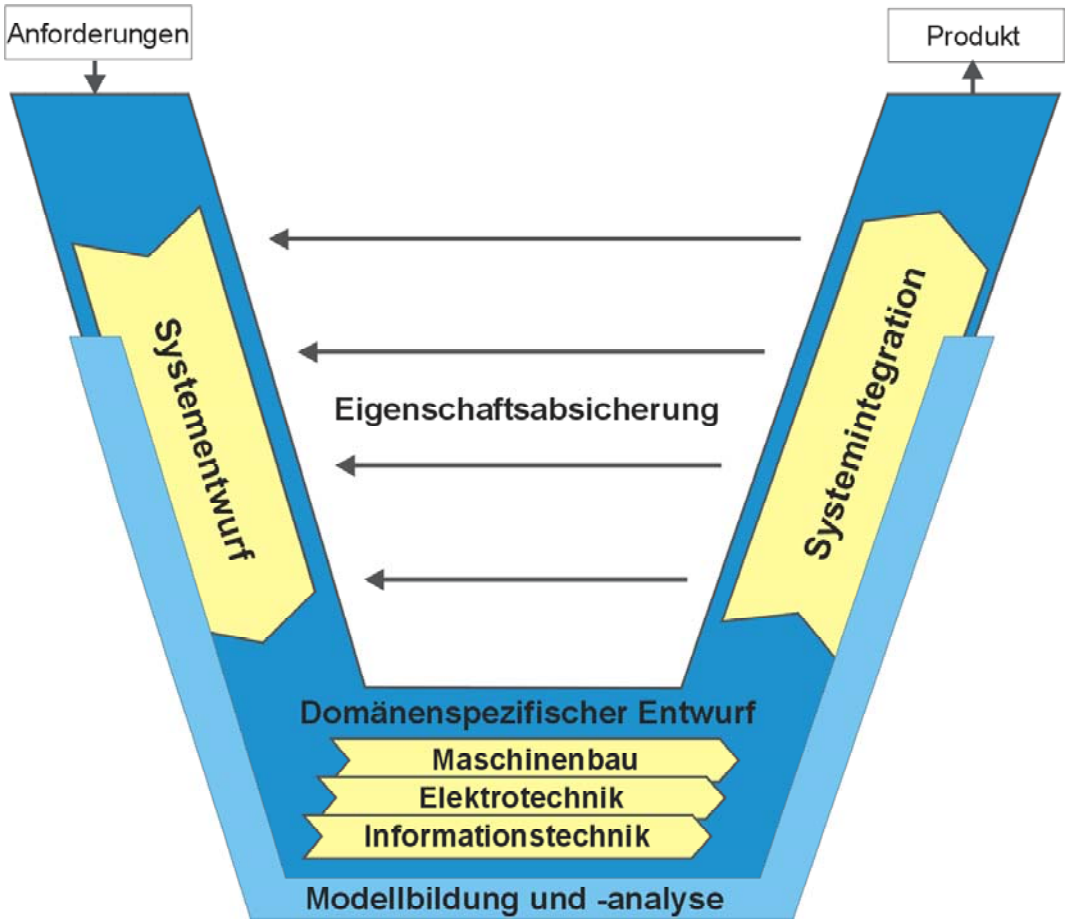


Abbildung 7: Das V-Modell als Makrozyklus (aus [VDI04])

Systementwurf: Der Systementwurf (vgl. Abbildung 8) beginnt mit dem Abstrahieren der in der Anforderungsliste beschriebenen Vorstellungen. Darauf aufbauen werden Funktions-, Wirk- und Baustrukturen erstellt und bewertet. Das Ergebnis ist ein disziplinübergreifendes Lösungskonzept, auch als Prinziplösung bezeichnet, welches Grundlage des domänenspezifischen Entwurfs ist.

Welche konkreten Schritte durchzuführen sind, um z.B. die Funktions- oder Wirkstruktur zu modellieren wird nur sehr oberflächig beschrieben. Für konkrete Angaben wird auf etablierte

Vorgehensweisen, wie z.B. die VDI-Richtlinie 2222 Blatt-1 (siehe Kapitel 3), hingewiesen. Das Ergebnis des Systementwurf ist das disziplinübergreifende Lösungskonzept[6].

Das hier beschriebene Vorgehen soll Vorfixierungen, die den möglichen Lösungsraum unnötig einschränken, vermeiden. Ziel ist es, das Wesentliche und Allgemeingültige der Problemstellung herauszuarbeiten und dabei helfen die Potentiale aller Disziplinen zu erkennen und zu nutzen.

Der Systementwurf wird als einer der wichtigsten Phasen während der Entwicklung angesehen.

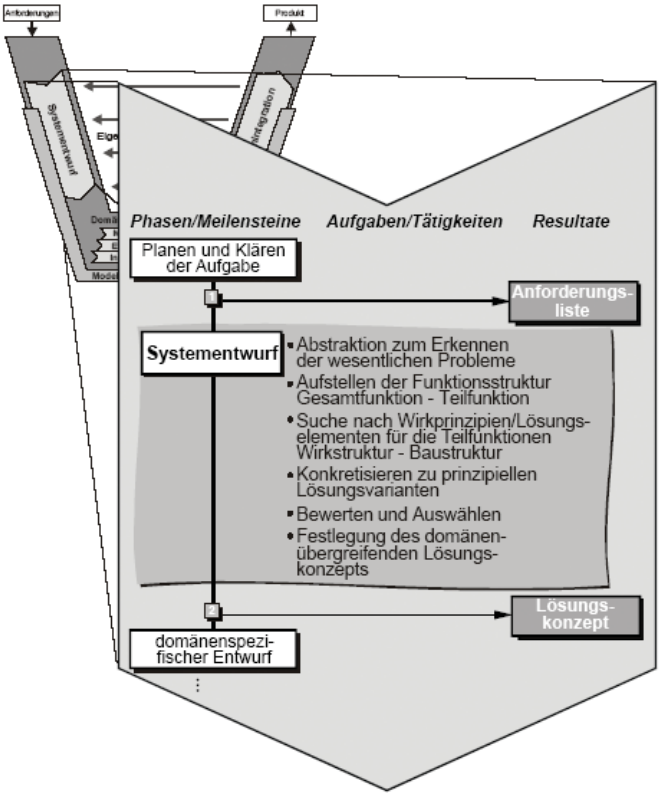


Abbildung 8: Tätigkeiten beim Systementwurf (aus [VDI04])

Domänenspezifischer[7] Entwurf: Auf Basis der Prinziplösung werden die zu entwickelnden Elemente auf die jeweiligen Disziplinen verteilt. Zur Entwicklung dieser Elemente wird in der Richtlinie allerdings kein konkretes Vorgehen beschrieben. Vielmehr wird auf die etablierten Vorgehensweisen der jeweiligen Disziplinen verwiesen (vgl. Kapitel 2.2.1).

Systemintegration: Die Systemintegration hat die Aufgabe die in den einzelnen Disziplinen entstandenen Elemente zu einem übergeordneten Ganzen (dem zukünftigen Produkt) zusammenzuführen. Als Integrationsarten werden *Integration verteilter Komponenten, Modulare Integration* und *Räumliche Integration* angegeben und grob beschrieben. Während dieser Integration erfolgt auch die Eigenschaftsabsicherung, d.h. das zusammengesetzte Produkt wird mit den Anforderungen verglichen und überprüft, ob es diese einhält. Hierzu werden verschiedene Techniken wie *Hardware-* und *Software-in-the-Loop* vorgeschlagen. Das Ergebnis der Systemintegration ist das fertige Produkt.

Einordnung

[6] Lösungskonzept wird auch als Prinziplösung bezeichnet
[7] Domäne und Disziplin werden in dieser Arbeit synonym betrachtet

Die VDI-Richtlinie ist ein erster Versuch die bisher unabhängig voneinander existierenden, disziplinspezifischen Vorgehensweisen aller drei Disziplinen zusammenzufassen. Sie stellt ein Rahmenwerk dar in das die disziplinspezifischen Vorgehensweisen eingebettet sind, wobei besonderer Wert auf die frühe Phase der Entwicklung, den Systementwurf, gelegt wird. Nur mit Hilfe dieser Richtlinie allein ist es nach Meinung des Autors jedoch nicht möglich ein mechatronisches System zu entwickeln.

2.2.2.3 Entwurf im Rahmen des SFB 614

Der Sonderforschungsbereich 614, „Selbstoptimierende Systeme des Maschinenbaus“ wurde im Jahr 2002 an der Universität Paderborn eingerichtet und beschäftigt sich mit der Frage, wie die Möglichkeiten der Informationstechnik noch intensiver genutzt werden können, um die Potentiale der Mechatronik weiter zu verbessern.

Künftige Systeme des Maschinenbaus werden aus Konfigurationen von Systemelementen mit einer inhärenten Teilintelligenz bestehen. Das Verhalten des Gesamtsystems wird durch die Kommunikation und Kooperation der intelligenten Systemelemente geprägt sein. Aus informationstechnischer Sicht handelt es sich um verteilte Systeme von miteinander kooperierenden Agenten [SFB01].

Es sei an dieser Stelle erwähnt, dass die Inhalte der vorliegenden Arbeit direkt im Rahmen der Forschungsaktivitäten im SFB 614 entstanden sind. Im folgenden soll daher nur der grobe Rahmen vorgestellt werden der nicht direkt Gegenstand der Arbeit ist.

Definition Selbstoptimierung

Der Begriff Selbstoptimierung wurde im Rahmen des SFB614 wie folgt definiert [SFB01]:

Unter Selbstoptimierung eines technischen Systems wird die endogene Änderung des Zielvektors auf veränderte Umweltbedingungen und die daraus resultierende zielkonforme autonome Anpassung der Struktur, des Verhaltens sowie der Parameter dieses Systems verstanden. Damit geht Selbstoptimierung über die bekannten Regel- und Adaptionsstrategien wesentlich hinaus; Selbstoptimierung ermöglicht handlungsfähige Systeme mit inhärenter „Intelligenz“, die in der Lage sind, selbständig und flexibel auf veränderte Umgebungsbedingungen zu reagieren.

Selbstoptimierende Systeme sind der Definition zufolge eine Erweiterung bekannter mechatronischer Systeme insbesondere um Informationstechnische Anteile.

Entwurfsprozess selbstoptimierender Systeme

Das strukturierte Vorgehen bei der Entwicklung selbstoptimierender Systeme ist von entscheidender Bedeutung. Nur so können die Tätigkeiten der Entwickler, die aus verschiedenen Disziplinen stammen, aufeinander abgestimmt und zu einem gemeinsamen Ziel geführt werden.

Im Rahmen des SFB 614 wurde Untersucht, ob und wie sich der Entwurfsprozess selbstoptimierender Systeme vom bekannten Entwurfsprozess für mechatronische Systeme unterscheidet, wobei die VDI-Richtlinie 2206 (vgl. 2.2.2.2) hier als Grundlage herangezogen wurde.

Innerhalb der ersten Förderperiode wurde ein sehr detaillierter Entwurfsprozess mit neun Phasen und insgesamt 165 einzelnen Prozessschritten definiert [SFB04]. Als Grundlage und Beispiel diente hierzu das vom Projekt „railcab“[8] [NBP] zur Verfügung gestellte Shuttle (vgl.

[8] RailCab, ehem. “Neue Bahntechnik Paderborn (NBP)“

Abbildung 9). Retrospektiv wurde ermittelt, wie die einzelnen Module des Shuttles entwickelt wurden, bzw. wie sie hätten entwickelt werden sollen.

Abbildung 9: Das Shuttle des railcab-Projektes

Die ermittelten neuen Phasen sind „Planen und Klären der Aufgabe“, „Konzipierung“, „Grobgestaltung“, „Reglerentwurf“, „Integration der Informationsverarbeitung“, „Virtuelles Prototyping“, „Prototyping“, „Ausarbeitung“ und „Fertigung“.

Während der Ermittlung des Entwurfsprozesses hat sich die in der Mechatronik bestehende Erkenntnis bestätigt, dass die wesentlichen Weichen in den frühen Phasen der Entwicklung gestellt werden. Im Rahmen des SFB 614 sind dies die Phasen „Planen und Klären der Aufgabe“ und „Konzipierung“. Am Ende der „Konzipierung“ steht das Konzept des Systems fest, d.h. die Prinziplösung ist spezifiziert auf deren Basis der disziplinspezifische Entwurf beginnen kann. Dies ist vergleichbar mit dem Systementwurfs der VDI-Richtlinie 2206.

Im Vergleich zur klassischen Vorgehensweise im Maschinenbau (nach VDI-Richtlinie 2222 [VDI97]) oder der Mechatronik (nach VDI-Richtlinie 2206 [VDI04]) gestaltet sich der Entwurf s.o. mechatronischer Systeme aufwendiger. Die Prinziplösung muss wesentlich mehr Aspekte - wie beispielsweise den Selbstoptimierungsprozess oder rekonfigurierbare Strukturen - des zu entwickelnden Systems betrachten [SFB04]. Die Beschreibung dieser Prinziplösung muss daher wesentlich umfangreicher sein.

Im Rahmen des SFB erfolgt sie mit Hilfe mehrerer, unterschiedlicher Modelle. Diese Modelle stellen jeweils unterschiedliche, spezifische Aspekte des Systems dar und werden daher als Partialmodelle bezeichnet. Diese Partialmodelle werden mit Hilfe unterschiedlicher Spezifikationstechniken modelliert, wobei sowohl neue, im SFB entstandene als auch bereits existierende Spezifikationstechniken verwendet werden. Die Erstellung der Partialmodelle erfolgt dabei nicht sequentiell, in einer zwangsläufigen Reihenfolge, sondern oftmals im Wechselspiel. Einige dieser Partialmodelle bauen aufeinander auf, andere entstehen parallel. Das Ergebnis ist ein kohärentes System von Partialmodellen (vgl. Abbildung 10).

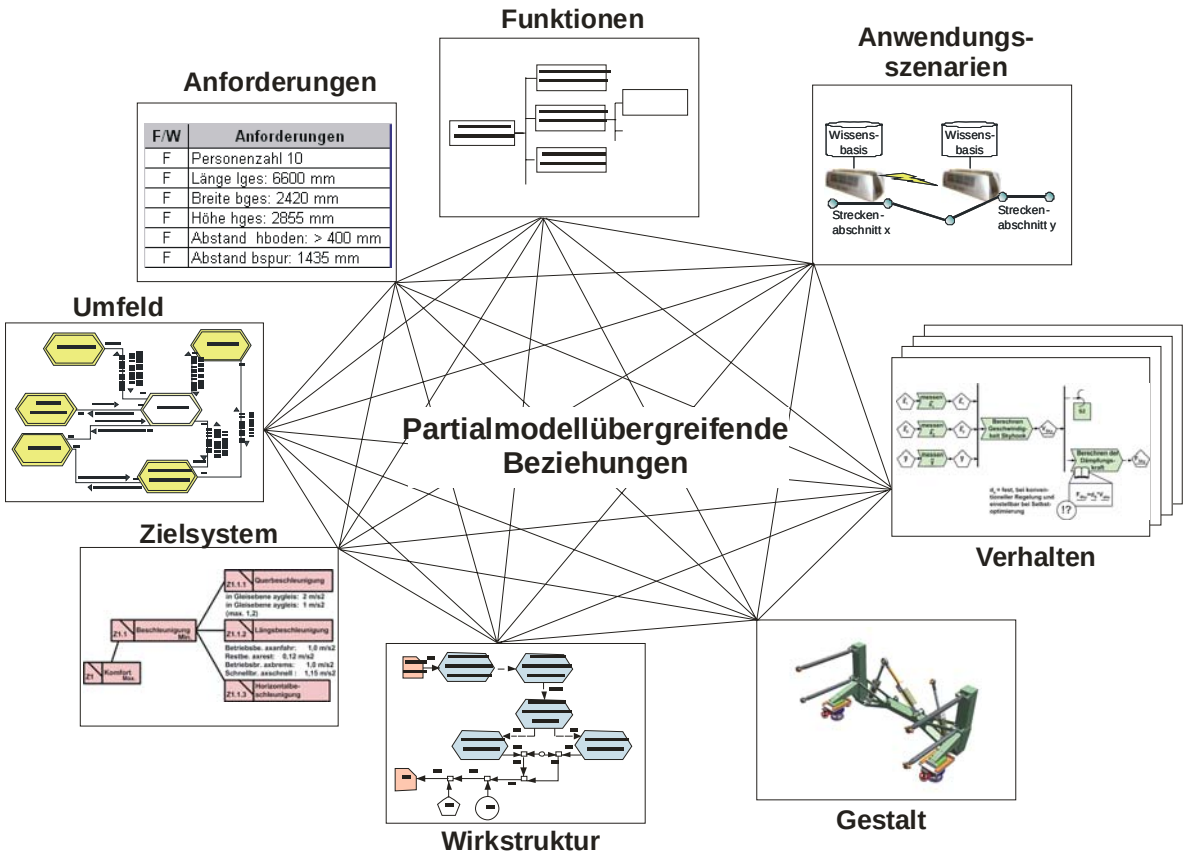


Abbildung 10: Partialmodelle zur Beschreibung der Prinziplösung s.o. Systeme

Zwei der genannten Spezifikationstechniken sind die Funktionen und die Wirkstruktur. Diese beiden Spezifikationstechniken werden in dieser Arbeit genauer betrachtet, da sie bereits seit langem im Entwurf mechatronischer Systeme eingesetzt werden und somit die meisten Erkenntnisse bzgl. ihrer Anforderungen und Abhängigkeiten bekannt sind.

Funktionen: Im Rahmen des SFB 614 wird zur Modellierung der Funktionen im Wesentlichen eine Funktionshierarchie verwendet. Die Modellierung einer Funktionsstruktur wird nicht vorgenommen. Welche konkreten Informationen die Funktionshierarchie enthält und wie diese Modelliert werden ist Gegenstand dieser Arbeit und wird in Kapitel 5 detailliert erklärt.

Wirkstruktur (Systemstruktur): Als Grundlage zur Modellierung der Wirkstruktur[9] dient die Notation nach Kallmeyer (vgl. 3.3). Mit ihrer Hilfe ist es möglich das zu entwickelnde System bzw. seine Elemente in Form einer vernetzten Struktur zu modellieren. Die Erweiterungen und Formalisierung dieser Notation ist Gegenstand dieser Arbeit und wird in Kapitel 6 detailliert erklärt.

Abhängigkeiten zwischen Funktionshierarchie und Systemstruktur: Die Systemstrukturmodellierung baut auf der Funktionsmodellierung (hier: Funktionshierarchie) auf. Die in der Funktionshierarchie beschriebenen Funktionen werden durch die in der Systemstruktur festgelegten Systemelemente realisiert. Jede Funktion muss durch Systemelemente erfüllt werden, damit das mechatronische System bzgl. der funktionalen Anforderungen vollständig beschrieben ist. Aus diesem Grund ist der Übergang von der Funktionshierarchie zur Systemstruktur ein entscheidender Schritt während des Systementwurfs. Dieser Übergang war bisher jedoch nur auf Basis der Erfahrung der Entwickler und weitestgehend ohne Rechnerunterstützung möglich. Diesen Übergang

[9] Im folgenden als Systemstruktur bezeichnet.

rechnerbasiert zu unterstützen ist daher Gegenstand dieser Arbeit und wird in den Kapiteln 7 und 8 erläutert.

Einordnung

Im Rahmen des SFB 614 wurde untersucht, welche Informationen benötigt werden, um ein selbstoptimierendes System zu entwickeln. Eine besondere Erkenntnis war, dass der Systementwurf eine entscheidende Rolle spielt. Aufgrund des s.o. Charakters dieser Systeme muss die spezifizierte Prinziplösung den Entwicklern der verschiedenen Disziplinen klar und verständlich sein. Mehr noch als bei bisher bekannten, mechatronischen Systemen müssen Aspekte wie Anforderungen, Struktur oder Verhalten genau festgelegt werden. Insbesondere weil s.o. Systeme sowohl ihre Struktur als auch ihr Verhalten selbstständig ändern können, muss der Rahmen in der diese Anpassungen durchgeführt werden können, genau vorgegeben werden.

Im SFB wurden daher unterschiedlicher Spezifikationstechniken definiert die dies ermöglichen. Es wurden sowohl neue Spezifikationstechniken entwickelt, als auch bereits bekannte erweitert. Darüber hinaus wurden die Zusammenhänge zwischen den Spezifikationstechniken betrachtet und informal beschrieben. Somit ist diese Form der Prinziplösung wesentlich detaillierter als es in den bereits vorgestellten VDI-Richtlinien der Fall ist.

2.2.2.4 Hardware/Software Codesign

Hardware / Software Codesign [Tei97] behandelt die integrative Entwicklung von Systemen die zum einen aus digitalen, mikroelektronischen Schaltung und zum anderen aus Software bestehen. Zu Beginn der Entwicklung wird das zu entwickelnde System umgangssprachlich beschrieben, wobei lediglich die Funktionalität spezifiziert und keine Angaben zur späteren Lösung gemacht werden. Daraufhin folgt die so genannte Partitionierung, bei der festgelegt wird, welche Disziplin die Lösung realisieren soll. Innerhalb der Disziplinen werden die Lösungen synthetisiert, d.h. sie können aus den Spezifikationen automatisch abgeleitet werden. Schließlich wird noch eine entsprechende Interfacesynthese durchgeführt um die Lösungen zu kombinieren (vgl. Abbildung 11).

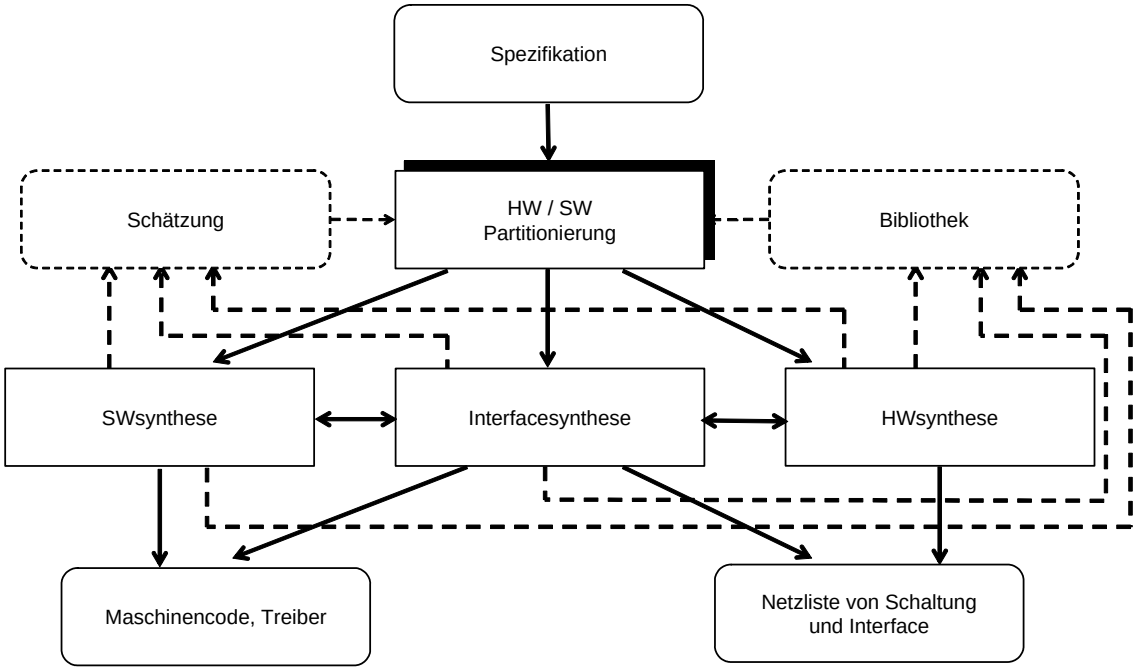


Abbildung 11: Vorgehen beim Hardware/Software Codesign [Tei97, S. 21]

Nach der Synthese werden dann umfangreiche Simulationen der Funktionalität und des Zeitverhaltens durchgeführt. Falls die Bewertung dieser Simulation Fehler aufweist ist ein Rücksprung bis hin zur Partitionierung möglich. Dort wird dann gegebenenfalls eine neue Partitionierung, also eine andere Verteilung der Aufgaben an die beiden Disziplinen durchgeführt.

Einordnung

Hardware / Software Codesign zeigt anhand der Disziplinen Mikroelektronik und Softwaretechnik, dass eine disziplinübergreifende Entwicklung möglich ist. Die beteiligten Disziplinen eignen sich hierfür recht gut, da die benötigten Voraussetzungen und realisierten Lösungswege nicht stark voneinander abweichen. Fraglich ist jedoch, ob dieses Vorgehen auch auf den Maschinenbau/Mechanik oder die Regelungstechnik anwendbar ist. Die Partitionierung zwischen Mechanik und Softwaretechnik beispielsweise ist deutlich schwieriger, da die Voraussetzungen und Realisierungen stark voneinander abweichen. Außerdem ist es möglich, dass erst durch die Kombination der Disziplinen eine geeignete Lösung realisiert werden kann. Eine Partitionierung wäre dann von Nachteil.

2.2.2.5 Invention Machine / TechOptimizer

Beim TechOptimizer handelt es sich um ein Softwaresystem der Firma Invention Maschine mit dessen Hilfe die methodische Konzeption disziplinübergreifender Systeme möglich ist. Die Grundlagen gehen dabei auf die Arbeiten von Altshuller zurück [Alt73], [Alt84]. Altshuller hat eine Methode entwickelt mit der es möglich ist technische Problemstellungen auf einen grundlegenden Widerspruch zurück zu führen, der sich dann in eine allgemeine Systematik einordnen lässt. Hierzu wird ein umfangreicher Patentkatalog zur Hilfe genommen, den Altshuller während seiner Arbeiten aufgestellt hat.

Der TechOptimizer erlaubt es dem Benutzer durch eine Vielzahl von ausführlich dokumentierten physikalischen Effekten zu navigieren, deren Vor- und Nachteile anzuschauen inkl. ihrer Abhängigkeiten. Auf diese Weise soll die Ideenfindung unterstützt werden.

Einordnung

Die Vorgehensweisen, die dem System zu Grunde liegen gleichen sehr stark den bekannten Vorgehensweisen anderer Ansätze aus dem Maschinenbau (z.B.: [VDI2221], [VDI2222]) und können daher problemlos in diese eingeordnet werden. Einen großen Vorteil bietet der TechOptimizer, indem er diese Vorgehensweisen mit Hilfe eines Werkzeugs entsprechend unterstützt. Leider ist die Benutzerführung noch nicht ausreichend genug an die Vorgehensweise angelehnt und lässt dem Benutzer zu viele Möglichkeiten. Darüber hinaus wird die Informationstechnik bisher nicht unterstützt. Es existiert kein adäquates Gegenstück das den physikalischen Effekten entsprechen würde.

Eingesetzt wird der TechOptimizer hauptsächlich während des Systementwurf. Allerdings gibt es keine festgelegte Vorgehensweise bzgl. seiner Nutzung, noch ist mit ihm die Modellierung von Wirkstrukturen/Systemstrukturen möglich.

2.2.2.6 Domänenübergreifende Konzeptionsumgebung für den Entwurf mechatronischer Systeme

Lippold stellt in seiner Arbeit [Lip00] ein Partitionierungsschema vor mit dessen Hilfe der Entwurf mechatronischer Systeme unterstützt wird. Dabei wird insbesondere auf den disziplinübergreifenden Charakter mechatronischer Systeme Rücksicht genommen, indem von bisherigen, disziplinspezifischen Ansätzen weitestgehend abstrahiert wird. Die Partitionierung erfolgt in fünf Stufen. Angefangen bei einem unpartitionierten

Konzeptionszustand bis hin zur disziplinfestlegenden Partitionierung. Innerhalb dieses Schemas werden die in Stufe 1 spezifizierten Elemente (Funktionen oder Wirkprinzipien) vorher festgelegten Partitionen zugeordnet. Diese Partitionen werden von Stufe zu Stufe verfeinert, bis eine Partition eindeutig einer Disziplin zugeordnet werden kann. Somit sind dann alle Elemente einer Disziplin zugeordnet und können im weiteren Verlauf des Prozesses Konkretisiert werden. Abbildung 12 zeigt das Partitionierungsschema.

Während der Partitionierung werden, von Lippold definierte Tätigkeitspaare iterativ durchlaufen. Dies sind:

- Synthese und Analyse
- Konkretisierung und Abstraktion
- Kreation und Simulation
- Darstellung und Erklärung
-

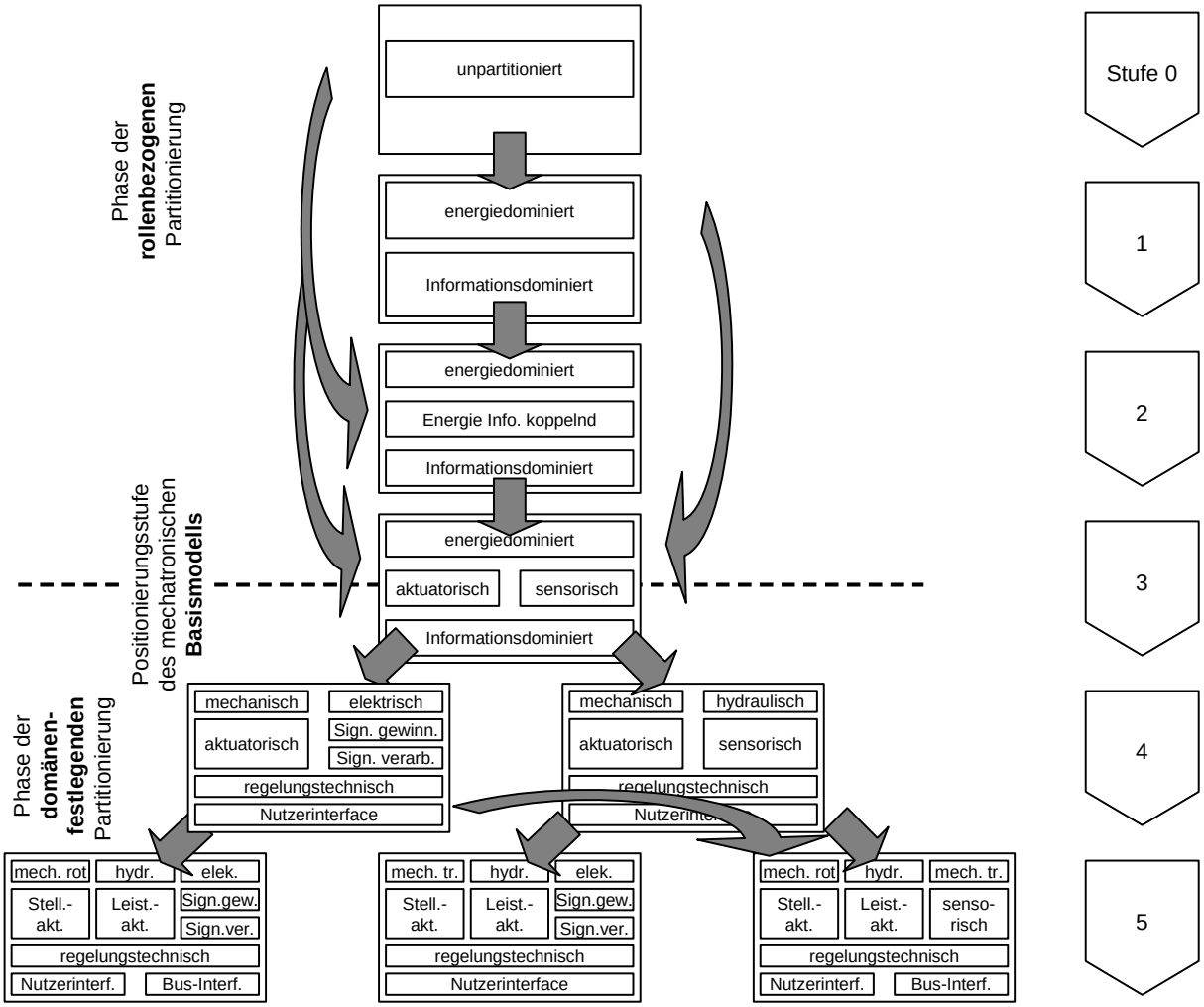


Abbildung 12: Partitionierung nach Lippold [Lip00, S. 78]

Die Ergebnisse dieser Tätigkeitspaare und damit der einzelnen Stufen im Partitionierungsschema werden durch unterschiedliche Modelle beschrieben:

- Mechatronische Funktionsebene[10]: Hier werden lösungsneutral die Systemfunktionen abgebildet. Die Verknüpfung der Funktionen erfolgt mit Hilfe der drei Flussarten Energie, Stoff und Information.
- Mechatronische Wirkebene[11]: Auf dieser Ebene wird der „verallgemeinerte Effekt“ eingeführt. Hierbei kann es sich sowohl um einen physikalischen wie auch logischen Effekt handeln. Letzterer integriert den informationstechnischen Teil in die Wirkstruktur.
- Mechatronische Verhaltensebene: Hier soll das Verhalten des Systems durch entsprechende Verhaltensmodelle und deren Simulation abgebildet werden.

Die Ergebnisse der Arbeit sind in die Konzeptionsumgebung ModCoDe (Modelling System for Conceptual Design) eingeflossen [WLB01].

Einordnung

Der Ansatz beschreibt den Entwurf mechatronischer Systeme mit Hilfe eines disziplinunabhängigen Partitionierungsschemas. Die in den VDI-Richtlinien 2221 und 2206 definierten Vorgehensweisen dienen als Grundlage und werden erweitert bzw. ersetzt.

Leider ist der Ansatz sehr abstrakt gehalten. Es ist nicht festgelegt welche Modelle (z.B.: Funktionsstruktur, Wirkstruktur) in welcher Stufe realisiert werden sollen. Eine konkrete Notation für die Funktions- oder Wirkstrukturmodellierung und ein entsprechender Übergang ist nicht definiert. Schließlich ist das präsentierte Beispiel eines Spulkopfes sehr knapp gehalten und zeigt auch nicht, wie ein solcher Spulkopf mit Hilfe des Partitionierungsschemas erstellt werden könnte.

2.3 Zusammenfassung

Zusammenfassend kann festgestellt werden, dass es erst wenige Vorgehensweisen gibt, die die disziplinübergreifende Entwicklung mechatronischer Systeme unterstützen. Erste Ansätze beruhen auf Entwicklungsprozessen einzelner Disziplinen wobei diese versuchen die Vorgehensweisen der anderen Disziplinen nachträglich hinzuzufügen. Dadurch werden diese Prozesse sehr stark von einer Disziplin dominiert und führen in der Regel nur unter hohem Zeit- und Kostenaufwand zum Ziel.

Jüngste Ansätze wie die VDI-Richtlinie 2206 oder die Arbeiten im Rahmen des SFB 614 zeigen neuen Vorgehensweisen auf die bereits von Beginn an versuchen alle Disziplinen in gleicher Weise zu berücksichtigen. Dabei steht insbesondere der Systementwurf im Vordergrund.

2.3.1 Besondere Bedeutung des Systementwurfs

Entscheidungen darüber, wie ein mechatronisches System einmal aussehen wird, welche Funktionalitäten es hat und nicht zuletzt was es kostet, wird zum großen Teil während des Systementwurfs entschieden. Die dort durchgeführten Arbeiten haben wesentlichen Anteil an dem Erfolg eines mechatronischen Systems [GBF+95]. Im Systementwurf werden Designentscheidungen und damit Kosten festgelegt die während der Konstruktion und Fertigung nur schwer zu ändern sind (vgl. Abbildung 13).

[10] entspricht der Funktionsstruktur gemäß VDI 2206
[11] entspricht der Wirkstruktur gemäß VDI 2206

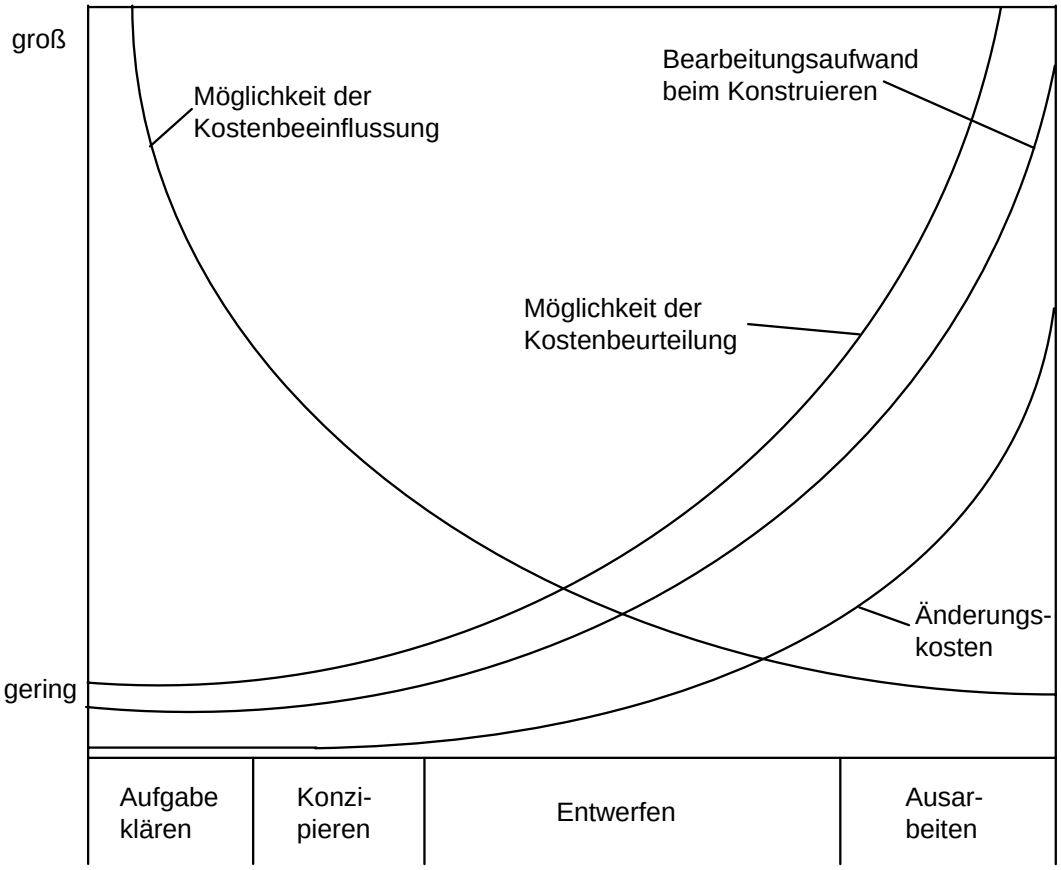


Abbildung 13: Möglichkeiten der Kostenbeeinflussung [Ehr95, S. 588]

Es ist daher besonders wichtig in dieser Phase die richtigen Entscheidungen zu fällen um ein kostspieliges Anpassen in einer späteren Phase zu vermeiden. Dies ist jedoch nur dann sinnvoll möglich, wenn das Problem komplett durchdrungen, verstanden und eine Lösung umfassend und verständlich spezifiziert wurde. *„Das Produktkonzept muss neben den reinen mechanischen Belangen auch die elektrischen, softwaretechnischen und regelungstechnischen Wirkungsweisen berücksichtigen“* [GEK01, S. 216].

Die meisten der in diesem Kapitel vorgestellten Ansätze konzentrieren sich auf die frühe Phase der Entwicklung, den Systementwurf, da die in dieser Phase entwickelten Modelle die Grundlage für die weitere Ausarbeitung darstellen.

Zwei Spezifikationstechniken, die während des Systementwurfs zum Einsatz kommen sind die Funktionshierarchie und die Wirk- bzw. Systemstruktur (vgl. 2.2.2). In der Funktionshierarchie werden die funktionalen Anforderungen an das mechatronische System spezifiziert und in der Systemstruktur werden die Systemelemente festgelegt, die diese Funktionen realisieren. Beide Spezifikationstechniken nehmen eine zentrale Rolle während des Systementwurfs ein, da durch sie systematisch festgelegt wird, aus welchem Elementen das zukünftige, mechatronische System aufgebaut sein soll.

Das Problem bei der Verwendung dieser Spezifikationstechniken ist jedoch, dass sie nur informal beschrieben sind, d.h. das insbesondere die Beziehungen und Abhängigkeiten untereinander unberücksichtigt bleiben. Dadurch entstehen Fehler in der Modellierung, die vorerst unentdeckt bleiben und möglicherweise erst während der Systemintegration zum Vorschein kommen. Ist dies der Fall, ist eine Korrektur meistens Zeit- und Kostenintensiv. Aus diesem Grund ist es sinnvoll, sowohl die beiden Spezifikationstechniken, als auch deren Beziehungen zu formalisieren und konsistent zu halten, um schon zu Entwurfszeiten mit Hilfe entsprechender Rechnerunterstützung mögliche Fehler zu vermeiden.

Eine vollständige Beschreibung eines mechatronischen Systems ist jedoch mit diesen beiden Spezifikationstechniken alleine noch nicht möglich. Hierfür werden noch weitere Spezifikationstechniken, z.B. zur Beschreibung des Verhaltens, benötigt.

2.3.2 Rechnerunterstützung

Die für den Systementwurf vorgeschlagenen Tätigkeiten, wie beispielsweise das Aufstellen von Funktions- und Wirkstrukturen, Zielsystemen oder Anwendungsszenarien, sind bisher nur sehr allgemein beschrieben. Insbesondere die Rechnerunterstützung wird noch nicht hinreichend genug berücksichtigt. Betrachtet man die Komplexität mechatronischer Systeme so muss man feststellen, dass ein enormes Wissen erforderlich ist, um für eine gegebene Aufgabenstellung ein System zu entwerfen, welches die bestmögliche Kombination der Möglichkeiten aller Disziplinen darstellt. Dies ist für einen einzelnen Entwickler annähernd unmöglich. Krause schreibt dazu: *„Der Produktentwickler kann nicht alles wissen“* [KJL04]. Dies zeigt, dass eine entsprechende Rechnerunterstützung während des Systementwurfs zunehmend wichtiger wird, wodurch sich die Komplexität des Entwurfs reduzieren lässt. Dadurch wird der Entwickler bei der Durchführung von Routineaufgaben entlastet wodurch mehr Zeit für kreative Tätigkeiten zur Verfügung steht.

KAPITEL 3: STAND DER TECHNIK

In Kapitel 2 ist beschrieben wie mechatronische Systeme aufgebaut sind, wie sie entwickelt werden und welche besondere Bedeutung der Systementwurf hat. Des Weiteren wurden die Funktions- und Wirkstrukturmodellierung als zentrale Tätigkeiten des Systementwurfs identifiziert.

Das folgende Kapitel beschreibt den Stand der Technik, der sich insbesondere mit der Funktions- und Wirkstrukturmodellierung beschäftigt. Beide Spezifikationstechniken kommen sowohl bei herkömmlichen mechatronischen Systemen, als auch bei selbstoptimierenden Systemen zum Einsatz. Aufgrund dieser Gemeinsamkeit stellen sie den konkreten Anwendungskontext dieser Arbeit darstellt.

3.1 VDI-Richtlinie 2222

Die VDI-Richtlinie 2222 - Blatt 1[12] [VDI97], ist als Verfeinerung der VDI-Richtlinie 2221[13] gedacht. Zweck der Richtlinie ist es *„Anleitung zu geben, wie der Prozess für das Finden der prinzipiellen Lösung eines technischen Produkts methodisch vollzogen und dokumentiert werden kann“*. Kern der VDI-Richtlinie 2222 Blatt-1 ist die Beschreibung des Vorgehens bei der Funktions- und Wirkstrukturmodellierung.

3.1.1 Funktionsmodellierung

Die Funktionsmodellierung der VDI-Richtlinie 2222 Blatt-1 setzt sich aus den zwei Schritten *Abstraktion der Gesamtfunktion* und dem *Aufstellen der Funktionsstruktur* zusammen.

Abstraktion der Gesamtfunktion: Aus der Problemspezifikation (Anforderungsliste) wird die Gesamtfunktion abgeleitet (z.B. „Wagen hochheben“). In der Regel ist diese jedoch sehr konkret und schränkt dadurch den möglichen Lösungsraum stark ein. Aus diesem Grund wird die Gesamtfunktion so weit abstrahiert, bis eine annähernd lösungsneutrale Beschreibung vorliegt. Abbildung 14 zeigt die schrittweise Abstraktion der Gesamtfunktion „Wagen hochheben“. Die Beschreibung einer Funktion erfolgt immer durch ein Objekt bzw. Substantiv (z.B. „Wagen“) und ein Prädikat bzw. Verb (z.B.: „hochheben“)[14].

[12] „Methodisches Entwickeln von Lösungsprinzipien“
[13] „Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte“
[14] Die Beschreibungsform wird auch als Subjektiv-Verb Beschreibungsform bezeichnet

Abstraktionsgrad		Objekt	Prädikat	Verlorengehende Zusatzinformationen	Ziel der Abstraktion
	Nr.	1	2	3	4
Niedrig	1	1.1 Wagen	1.2 hochheben	---	---
	2	2.1 festen Körper	2.2 heben	2.3 Wagen vom Boden hochheben	2.3 Verallgemeinerung des Vorgangs
	3	3.1 fester Stoff	3.2 bewegen	3.3 Zusammenhängenden Körper entgegengesetzt zur Schwerkraft bewegen	3.4 Zuordnungsmöglichkeit zu Sollfunktionen der physikalischen Funktionsstrukturen
Hoch	4	4.1 Stoff	4.2 leiten	3.4 Art des Stoffes, Art des Transports	4.4 Zuordnungsmöglichkeit zu Sollfunktionen der Allgemeinen Funktionsstruktur

Abbildung 14: Schrittweise Abstraktion der Gesamtfunktion (aus [VDI97])

Die Herleitung der abstrakten Gesamtfunktion reicht jedoch nicht aus um alle Anforderungen der Anforderungsliste zu erfüllen. Die Gesamtfunktion wird daher nicht nur in genau eine abstrakte Funktion zerlegt, sondern in mehrere. Dies geschieht Schritt für Schritt auf mehreren Abstraktionsebene, bis die größtmögliche Abstraktion erreicht ist (vgl. Abbildung 15).

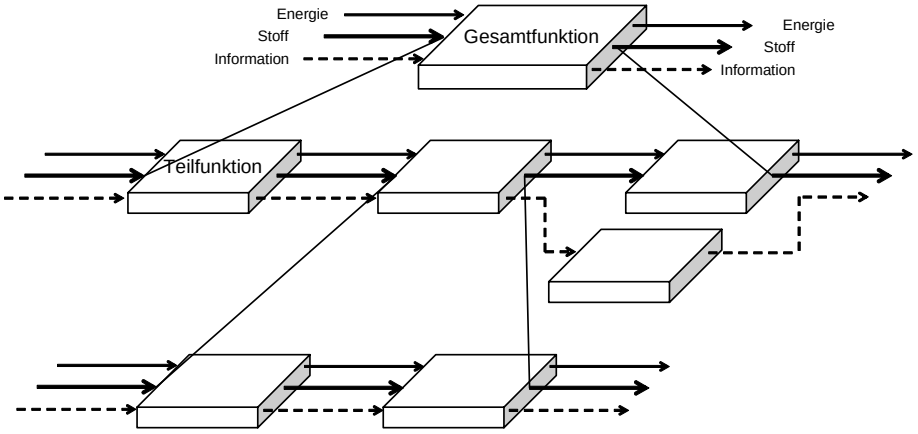


Abbildung 15: Funktionszerlegung nach Pahl/Beitz [PB03, S. 43]

Für die Beschreibung abstrakter Funktionen wurden zwei Postulate aufgestellt:

- „Die Funktionen aller Maschinen, Geräte und Apparate lassen sich mit Hilfe der so genannten drei „Allgemeinen Größen; *Stoff, Energie, Information*“ beschreiben“.
- „Die notwendigen und hinreichenden Zustandsänderungen sind: *Speichern, Leiten, Umformen, Wandeln, Verknüpfen“*.

Durch die Kombination der allgemeinen Größen und der Zustandsänderungen werden die so genannten *Allgemeinen Funktionen* gebildet. Insgesamt lassen sich die drei allgemeinen Größen und die fünf Zustandsänderungen zu insgesamt 30 unterschiedlichen *Allgemeine Funktionen* kombinieren.

Aufstellen der Funktionsstruktur: Nachdem nun die Gesamtfunktion abstrahiert und zusätzliche Funktionen spezifiziert wurden, werden diese mit Hilfe verschiedener Flussarten miteinander verknüpft. Bei den Flussarten handelt es sich um Flüsse der allgemeinen Größen, *Stoff, Energie* und *Information.* Zur graphischen Darstellung *Allgemeiner Funktionsstrukturen* wurde von Roth [Rot00] eine entsprechende Notation entwickelt (vgl. Abbildung 16).

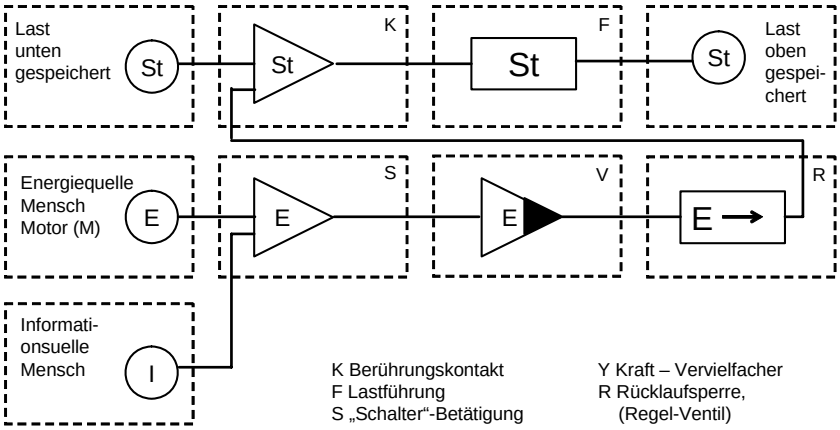


Abbildung 16: Funktionsstruktur der Hauptfunktion Stoff speichern (aus [VDI 97])

3.1.2 Wirkstrukturmodellierung

Nachdem die Funktionsstruktur erstellt wurde, muss nun für jede Funktion eine entsprechende Lösung gesucht werden. Zu diesem Zweck wird empfohlen sich für jede Funktion mehrere mögliche Lösungen zu überlegen. Diese Lösungen werden dann in ein Ordnungsschema einsortiert. D.h. für jede „Allgemeine Funktion" werden alle, dem Entwickler bekannten Lösungen eingetragen. Auf diese Weise entsteht die in Abbildung 17 gezeigte Matrix.

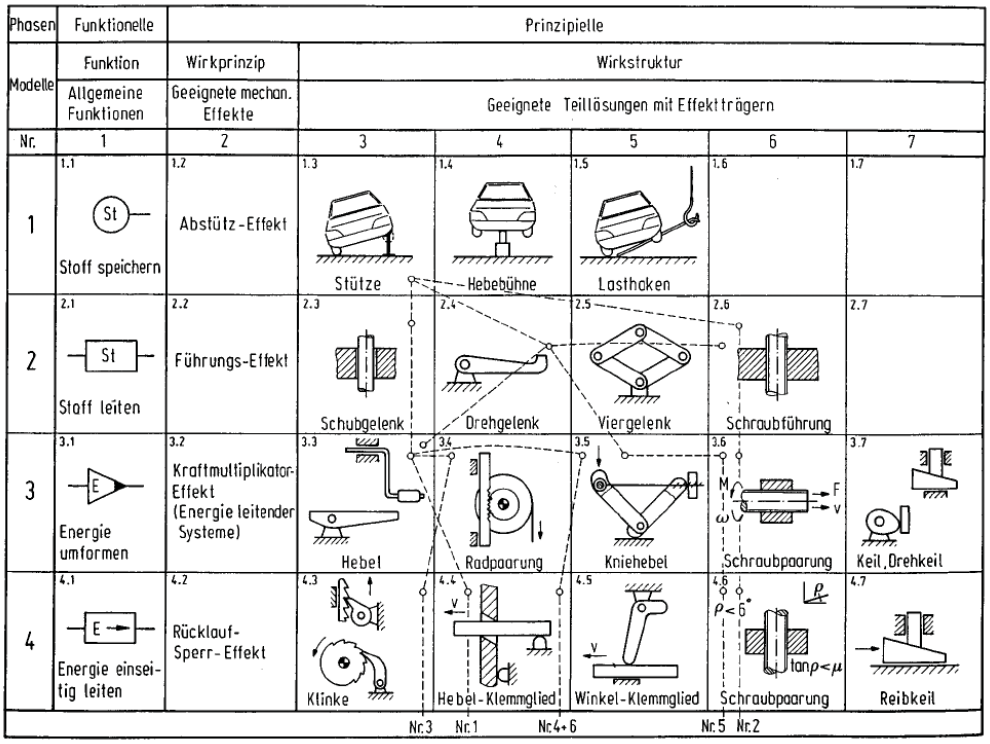


Abbildung 17: Allgemeine Funktionen und entsprechend zugeordnete Lösungen

Auf Grundlage dieser Matrix wird dann für alle „Allgemeinen Funktionen“ jeweils eine Lösung ausgewählt. Durch die Kombination der gewählten Lösungen entsteht die Wirkstruktur (gestrichelte Linien in Abbildung 17). Auf diese Weise können auch mehrere Wirkstrukturen ermittelt werden, von denen dann am Ende eine verwirklicht wird. Die Methode zum Befüllen der Matrix und Auswählen von Lösungen wird als „Morphologischer Kasten“[15] [Zwi71] bezeichnet.

Durch weitere Ergänzungen, wie z.B. Skizzen oder überschlägige Berechnungen ergibt sich schließlich die Prinzipiösung.

Die Durchführung der beschriebene Vorgehensweise erfolgt manuell durch den Entwickler. Basierend auf seinen Erfahrungen und seinem Wissen wird sowohl die Funktionshierarchie/-struktur, als auch der morphologische Kastens aufgestellt. Ob die gewählten Lösungen alle Funktionen erfüllen, zueinander kompatibel sind und möglicherweise von anderen Lösungen abhängen, weiß nur der Entwickler selbst. Bei auftretenden Änderungen (z.B.: Anforderungsänderung) muss er diese Änderungen integrieren und die Modelle konsistent zueinander halten.

3.2 Funktions- und Wirkstrukturmodellierung (Pahl/Beitz)

Die Funktionsstrukturmodellierung nach Pahl/Beitz [PB03] gleicht im Wesentlichen der Modellierung gemäß der VDI-Richtlinie 2222 Blatt-1. Der wesentliche Unterschied liegt in der Beschreibung der Funktionen.

3.2.1 Funktionsmodellierung

Während nach der VDI-Richtlinie 2222 Blatt-1 nur fünf Zustandsänderungen (Verben) und drei allgemeine Größen (Substantive) zugelassen sind, ist diese Einschränkung nach Pahl/Beitz nicht gegeben. Hier können die Substantive und Verben beliebig gewählt werden (vgl. Abbildung 18). Der Grund hierfür ist, dass unterschiedliche Entwickler unterschiedliche Vorstellungen und Erfahrungen besitzen und aus diesem Grund Funktionen auf verschiedenen Abstraktionsebenen (z.B.: „Öl pumpen“ oder „Flüssigkeit fördern“) beschreiben. Durch diesen Ansatz wird eine größere Flexibilität ermöglicht. Eine Sammlung von möglichen Verben wurde von Birkhofer [Bir80] aufgestellt und kann als Referenzliste betrachtet werden. Eine Sammlung von Substantiven wird in der Arbeit von Pahl/Beitz nicht vorgeschlagen.

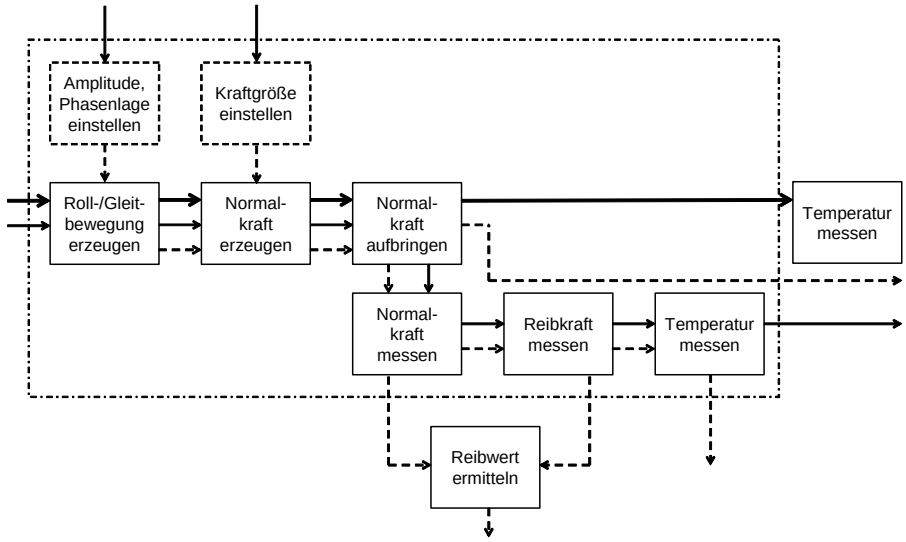


Abbildung 18: Funktionsstruktur nach Pahl/Beitz [PB03, S. 229]

[15] „Morphologischer Kasten“: Eine Entwickelung des Schweizer Astrophysikers Fritz Zwicky (1898-1974).

Die Aufgliederung der Gesamtfunktion in Teilfunktionen und die Verknüpfung der Funktionen mit Hilfe der drei allgemeinen Größen Stoff, Energie und Information zu einer Funktionsstruktur gleicht der der VDI-Richtlinie 2222 Blatt-1. Eine solche Funktionsstruktur wird auch *spezielle Funktionsstruktur* genannt. Als Notation wird eine Blockdarstellung gewählt.

Wie auch schon bei der Vorgehensweise nach der VDI-Richtlinie 2222, erfolgt auch nach Pahl/Beitz die Modellierung ausschließlich von Hand.

3.2.2 Wirkstrukturmodellierung

Die Wirkstrukturmodellierung nach Pahl/Beitz ist identisch mit der der VDI-Richtlinie 2222 Blatt-1. Auch hier wird der morphologische Kasten verwendet.

3.3 Modellierung der Prinziplösung (Kallmeyer)

In der Dissertation von Ferdinand Kallmeyer [Kal98] wird eine in einen übergeordneten entwicklungsmethodischen Gesamtrahmen eingebettete Methode zur Modellierung der Prinziplösung mechatronischer Systeme vorgestellt. Der Schwerpunkt der Arbeit liegt auf der Beschreibung und methodischen Anwendung von Konstrukten zur disziplinübergreifenden Konzipierung mechatronischer Systeme.

Die Eigenschaften der Methode können wie folgt zusammengefasst werden:

- Die Methode ist eingebettet in einen entwicklungsmethodischen Gesamtrahmen.
- Mit Hilfe einer neuen Notation ist es möglich Wirkstrukturen zu modellieren.
- Ein definiertes Vorgehen erlaubt, ausgehend von einer funktionalen Beschreibung zur Wirkstruktur und zur Prinziplösung zu gelangen.

3.3.1 Funktionsmodellierung

Die Funktionsmodellierung nach Kallmeyer gliedert sich in die *Funktionshierarchie* und *Funktionsstruktur*.

Funktionshierarchie: Zu Beginn der Funktionsmodellierung wird die Gesamtfunktion bestimmt. Diese wird dann in weitere Teilfunktionen zerlegt. Kallmeyer spricht dabei von der „funktionalen Dekomposition“. Das Ergebnis dieser Dekomposition ist die Funktionshierarchie. Abbildung 19 zeigt eine vereinfachte Funktionshierarchie für die Funktion „Bremsen (mit ABS)“.

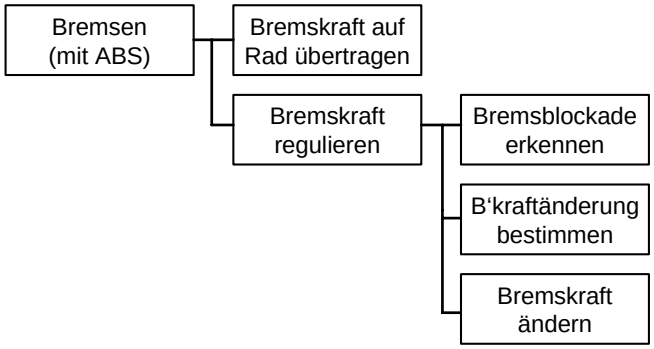


Abbildung 19: Funktionshierarchie der Gesamtfunktion „Bremsen (mit ABS)“

In der Funktionshierarchie repräsentiert ein Rechteck jeweils eine Funktion des Systems. Die Kanten zwischen den Funktionen stehen für die Dekomposition und lassen sich – jeweils von der übergeordneten Funktion aus – als ”besteht aus“ lesen.

Für die Beschreibung der Funktionen werden keine konkreten Vorgaben gemacht, d.h. die Funktionsbeschreibung kann beliebig sein.

Funktionsstruktur: Im zweiten Schritt der Funktionsmodellierung werden die Teilfunktionen der Funktionshierarchie mit Hilfe von Stoff-, Energie- und Informationsflüssen erknüpft. Hierbei werden nur die Funktionen berücksichtigt, die selber nicht mehr weiter dekomponiert wurden (Blätter der Funktionshierarchie).

Das Ergebnis der Flussverkettung ist eine Funktionsstruktur nach dem Black-Box-Prinzip, wie sie auch Pahl/Beitz verwenden (vgl. 3.2.1). Die Funktionsstruktur des ABS-Beispiels ist in Abbildung 20 dargestellt. An den Flüssen kann ein erklärender Text annotiert werden, der dem besseren Verständnis dienen soll.

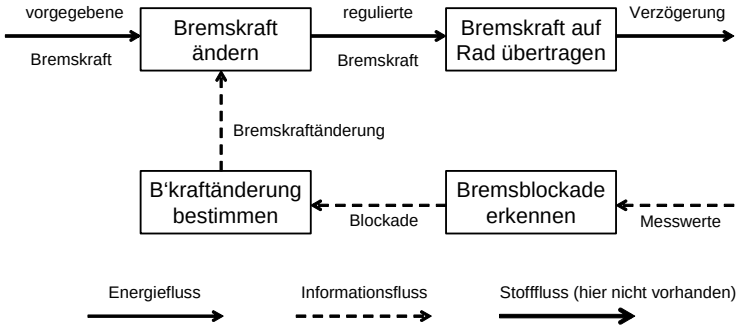


Abbildung 20: Funktionsstruktur der Funktion "Bremsen (mit ABS)"

3.3.2 Wirkstrukturmodellierung

Im Anschluss an die Funktionsmodellierung folgt die Wirkstrukturmodellierung. Zur Modellierung der Wirkstruktur hat Kallmeyer eine eigene Notation definiert.

Notation zur Beschreibung der Wirkstruktur

Kallmeyer bezeichnet die mit Hilfe seiner Notation erstellten Struktur nicht Wirkstruktur, sondern Systemstruktur, wodurch er sich von der bekannten Bezeichnung abgrenzt. Bestandteile der Systemstruktur sind zum einen Wirkprinzipien und zum anderen Lösungselemente, die zusammenfassend als Systemelemente bezeichnet werden.

Wirkprinzipien werden durch ein Oval notiert, Lösungselemente durch ein liegendes Sechseck und eine Menge wählbarer Lösungselemente durch ein liegendes Sechseck mit doppelter Umrandung. Zur besseren Strukturierung dienen so genannte Aggregationen, die eine Anzahl weiterer Elemente enthalten und sie somit zu einer Gruppe zusammenfassen. Abbildung 21 zeigt die verschiedenen Systemelemente.

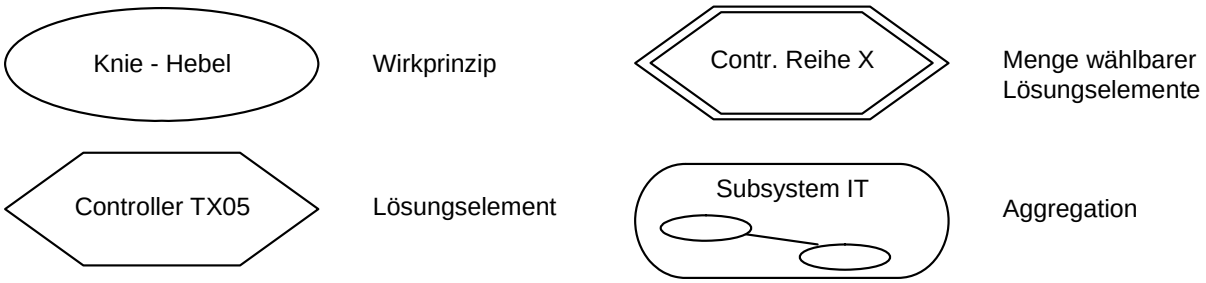


Abbildung 21: Systemelemente der Wirkstruktur

Die Verknüpfung der verschiedenen Systemelemente erfolgt mit Hilfe von Assoziationen. Assoziationen zwischen Systemelementen spiegeln die funktionelle Struktur des Systems wieder. Das heißt, Systemelemente, die direkt zusammenarbeiten oder kommunizieren, werden mit Assoziationen verknüpft. Assoziationen werden in der Systemstruktur als gerichtete oder ungerichtete Kanten notiert (vgl. Abbildung 22).

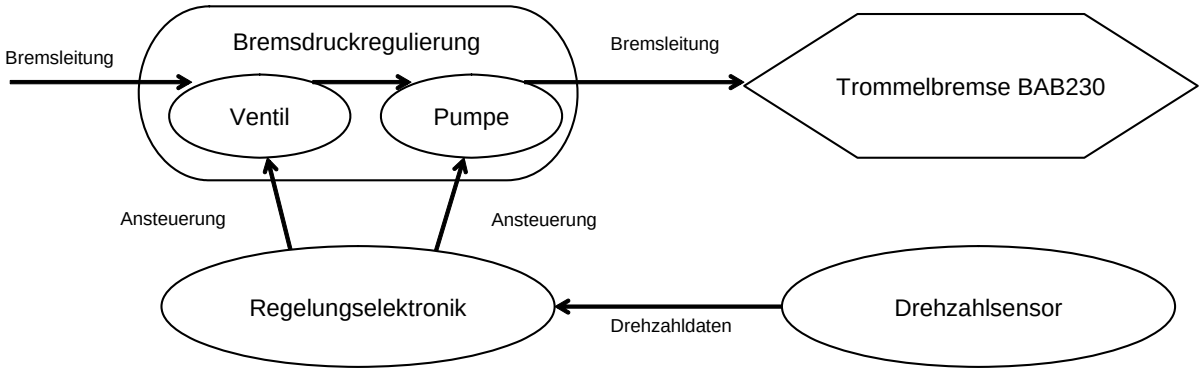


Abbildung 22: Systemstruktur für das ABS-Bremssystem

Kallmeyer leitet die Assoziationen von den Flussbeziehungen aus der Funktionsstruktur ab. Er übernimmt sie jedoch nicht direkt, sondern abstrahiert von einigen Informationen. So wird beispielsweise die Flussart nicht übernommen; mehrere Flüsse werden zu einer Assoziation zusammengefasst.

Übergang von der Funktionsstruktur zur Systemstruktur

Für die Suche nach Wirkprinzipien bzw. Lösungselementen gibt Kallmeyer in seiner Arbeit keine genaue Vorgehensweise an. Er sagt lediglich, dass eine Zuordnung stattfindet.

Als Resultat der Suche ergibt sich eine Zuordnung von Systemelementen zu den Teilfunktionen. Dabei können einer Teilfunktion durchaus mehrere, verschiedene Systemelemente zugeordnet sein, insbesondere wenn ein Systemelement allein die Teilfunktion nicht erfüllen kann.

Prinziplösung

Das Aufstellen der Systemstruktur alleine reicht jedoch nicht aus um das Konzept hinreichend genug zu beschreiben. Im Anschluss müssen noch weitere Informationen (Skizzen, Berechnungen, etc.) den Systemelementen zugeordnet werden. Durch diese Zuordnung entsteht dann die Prinziplösung. Die Prinziplösung nach Kallmeyer besteht demzufolge aus der Systemstruktur und zusätzlichen Informationen die den Systemelementen zugeordnet sind (vgl. Abbildung 23).

Kallmeyer beschreibt in seiner Arbeit erstmalig eine Notation zur Modellierung der Systemstruktur. Allerdings erfolgt auch bei ihm die Modellierung der Funktionshierarchie, der Funktionsstruktur und der Systemstruktur ausschließlich manuell. Keine der Spezifikationstechniken wird formal beschrieben, noch wird gezeigt, wie die Konsistenz zwischen ihnen sichergestellt werden kann.

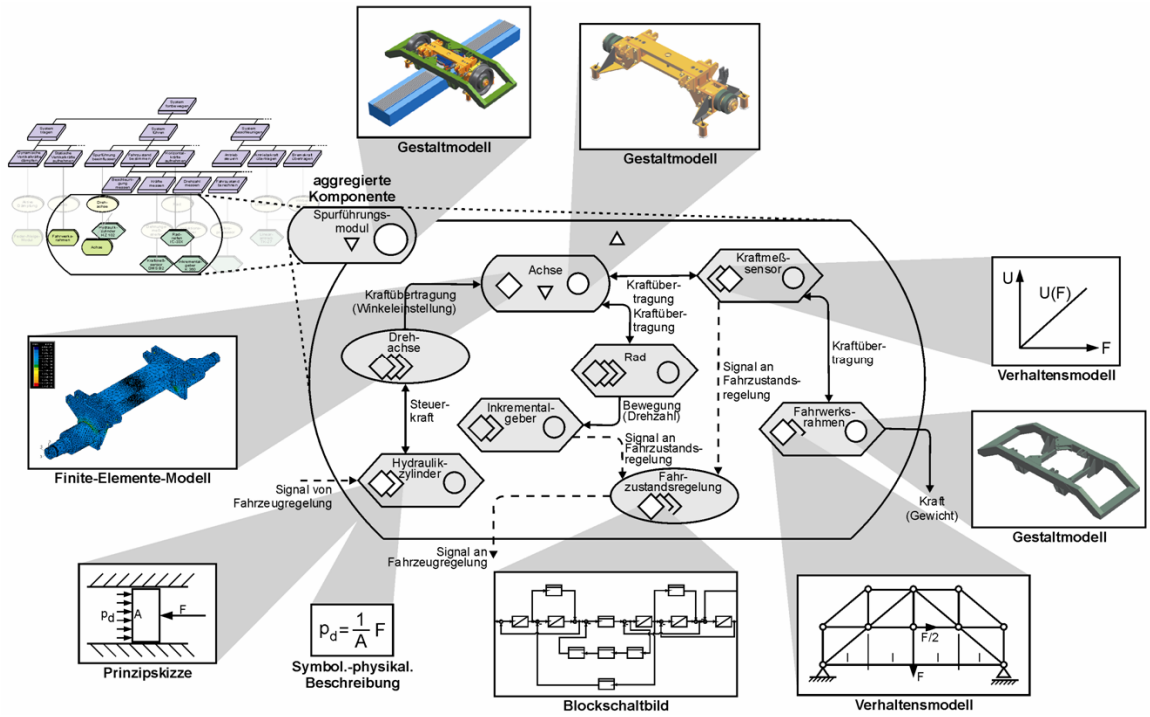


Abbildung 23: Prinziplösung nach Kallmeyer

3.4 Konzipierung mechatronischer Produkte (Flath)

In der Arbeit von Martin Flath [Fla01] wird ein Ansatz vorgestellt, der die Wirk- bzw. Systemstrukturmodellierung nach Kallmeyer erweitert.

3.4.1 Wirkstrukturmodellierung

Um eine vollständige Wirkstruktur zu modellieren reichten die bisherigen Ansätze nicht aus. Insbesondere sind Beschreibungsmöglichkeiten von Zuständen bzw. Störzuständen in die Wirkstruktur zu integrieren. Zu diesem Zweck wurde die Systemstruktur von Kallmeyer aufgegriffen und um entsprechende Konstrukte ergänzt. Zustände bzw. Störzustände können mit Hilfe von Zustandsparametern genauer spezifiziert werden. Die genauen Konstrukte sind in Abbildung 25 dargestellt.

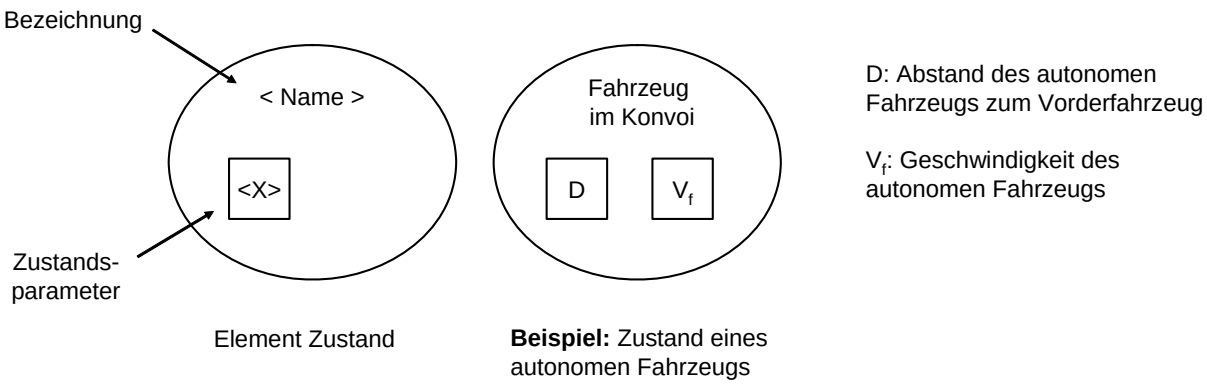


Abbildung 24: Zustände nach Flath

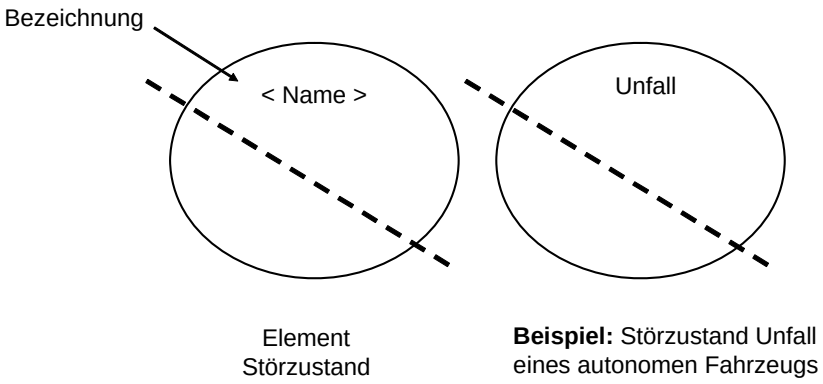


Abbildung 25: Störzustände nach Flath [Fla01, S. 90/91]

Zusätzlich zur Erweiterung der Notation wurde eine Vorgehensweise zur Erstellung der Wirkstruktur definiert. Diese Vorgehensweise gleicht der der VDI-Richtlinie 2222 Blatt-1, ergänzt diese aber um spezifische Aspekte bezüglich der Zustandsermittlung.

3.5 Projekt „iViP“

Im Rahmen des Verbundprojektes „integrierte virtuelle Produktentstehung“ (iViP02) ist im Teilprojekt 3.2 ein Software-Prototyp entwickelt worden, der ein funktionsorientiertes Entwerfen mechatronischer Produkte ermöglicht. Der entwickelte Prototyp besteht aus vier eigenständigen Editoren für Anforderungen, Funktionen, Baustrukturen und Constraints. Von besonderer Bedeutung für diese Arbeit ist der Funktionseditor.

3.5.1 Funktionsmodellierung

Der Funktionseditor ermöglicht die Modellierung und Visualisierung sowohl von Funktionshierarchien als auch Funktionsstrukturen. Darüber hinaus bietet der Editor die Möglichkeit an, einzelne Funktionsbeschreibungen in einer Bibliothek zu speichern.

Bei der Darstellung einer Funktionsstruktur wird eine Funktion in dem Funktionseditor durch ein Rechteck dargestellt. Eine Funktion ist definiert als eine Transformation der drei allgemeinen Größen der Mechatronik (Energie, Stoff, Information). Sie wird beschrieben durch verschiedene Ein- bzw. Ausgangsparameter über die die Funktionen verbunden werden.

Die Darstellung der Funktionshierarchie wird hauptsächlich als Navigationsmittel genutzt, die eigentliche Funktionsbeschreibung erfolgt in der Funktionsstruktur.

Zur Beschreibung der Funktionen müssen bereits alle Ein- bzw. Ausgangsparameter spezifiziert werden, damit die Funktionen verbunden werden können. Dies bedeutet jedoch, dass schon bei der Erstellung der Funktionsstruktur eine genaue Vorstellung vorliegen muss, mithilfe welcher Systemelemente die Funktion später realisiert werden soll. Somit ist eine nahezu lösungsneutrale Darstellung mit Hilfe des Funktionseditors nicht möglich.

Weiterhin ist festzuhalten, dass der Funktionseditor *„faktisch nur mechanische Aspekte zur Integration von Modelldaten behandelt“* [CZ03]. Ein Beispiel für den sehr mechanisch orientierten Aspekt im Funktionsdesigner ist die Möglichkeit, Zeichnungen von Computer Aided Design (CAD) Programmen zu hinterlegen und später wieder abzurufen. Für die mechanische Entwicklung mag dies noch sehr vorteilhaft erscheinen, jedoch werden hierdurch schon sehr konkrete Lösungen verlangt und benutzt, um eine Funktionsstruktur aufzubauen. Dadurch ist eine ganzheitliche und interdisziplinäre Repräsentation eines mechatronischen Produktes nicht mehr gegeben.

3.6 Funktionsstrukturentwicklung innovativer Produkte (Langlotz)

Langlotz beschreibt in seiner Arbeit [Lan00] einen Ansatz zur Modellierung zweier unterschiedlicher Funktionsstrukturen, die „allgemeine Funktionsstruktur“ und die „spezielle Funktionsstruktur“.

3.6.1 Funktionsmodellierung

Die allgemeine Funktionsstruktur beruht auf der Funktionsstrukturmodellierung der VDI-Richtlinie 2222 Blatt-1 (vgl. 3.1.1). Die spezielle Funktionsstruktur ist angelehnt an die Arbeiten von Pahl/Beitz (vgl. 3.2.1).

Die spezielle Funktionsstruktur erweitert die allgemeine Funktionsstruktur, indem zum einen mehr Verben und Substantive zur Verfügung stehen und zum anderen Ein- bzw. Ausgabeparameter spezifiziert werden können. Beide Funktionsstrukturen können ineinander überführt werden. Aufgrund der Vielfalt der möglichen Verben und Subjektive zur Beschreibung der speziellen Funktionsstruktur hat Langlotz eine Reihe von Taxonomien aufgestellt um diese Vielfalt in den Griff zu bekommen.

In seinem Ansatz gibt Langlotz an, dass die Erstellung der Funktionsstrukturen zusammen mit der Erstellung von Funktionshierarchien erfolgt. Hierzu definiert er sowohl eine Top-Down Variante zur Funktionshierarchiemodellierung (ähnlich Kallmeyer, vgl. 3.3.1) als auch eine Bottom-Up Variante.

Die Arbeit unterstützt den Systementwurf mechanischer bzw. mechatronischer Produkte durch Erweiterungen im Bereich der Funktionsstrukturmodellierung. Insbesondere die aufgestellten Taxonomien für Verben und Substantive sind sehr hilfreich und tragen zum besseren Verständnis und leichteren Handhabung bei. Der Übergang von der allgemeinen Funktionsstruktur zur speziellen Funktionsstruktur wird nur angesprochen, jedoch nicht genau spezifiziert. Wie die einzelnen Funktionen überführt werden, bleibt offen.

Des Weiteren wird nicht angegeben, wie aus den Funktionsstrukturen eine Wirk- oder Systemstruktur ermittelt werden kann. Dieser Übergang fehlt.

3.7 Semantische Funktionsmodellierung (Puri)

Der Ansatz von Puri [Pur03] beschäftigt sich mit der Problematik der natürlichen Sprache im Rahmen der Funktionsmodellierung.

3.7.1 Funktionsmodellierung

Funktionen werden durch die Kombination eines Substantivs und eines Verbs spezifiziert. Grundlage hierbei sind die allgemeinen Funktionen (vgl. 3.1.1). Diese zu kombinieren reicht jedoch nicht aus um Funktionen zu beschreiben, weshalb die speziellen Funktionsstrukturen definiert wurden (vgl. 3.2.1).

Um Funktionen jedoch vergleichbar zu machen, ist eine eindeutig Beschreibung notwendig. Hier hat Puri verschiedene Taxonomien sowohl für Substantive als auch für Verben definiert. Abbildung 26 zeigt den Auszug der Taxonomie des Verbs „ändern“.

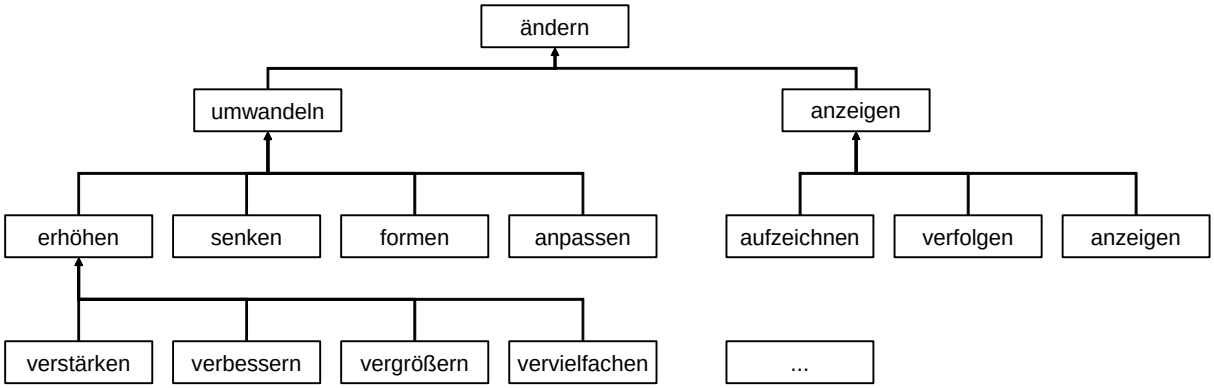


Abbildung 26: Taxonomie des Verbs "ändern" nach Puri [Pur03, S. 88]

Die entstehenden Funktionshierarchien oder Funktionsstrukturen können dann mit Hilfe der FMEA[16] optimiert werden.

Zur Rechnerunterstützung wurde das Softwarewerkzeug „KompAss“ entwickelt. Mit diesem Werkzeug ist es möglich verschiedene Taxonomien zu modellieren und zu verwalten.

Der Ansatz von Puri greift das Problem der Mehrdeutigkeit im Rahmen der Funktionsmodellierung auf. Diesem Problem wird mit Hilfe von verschiedenen Taxonomien begegnet. Auf diese Weise wird jeder Entwickler in die Lage versetzt Funktionen nach seinem eigenen Ermessen zu beschreiben, wobei die Vergleichbarkeit der Funktionen erhalten bleibt.

Abgesehen von der Subjektiv-Verb Beschreibung ist es nicht möglich weitere Angaben zu einer Funktion zu machen. So kann z.B. keine zusätzliche Information wie „Gesamtmasse kleiner 100 kg“ modelliert werden. Auch die Flussbeziehungen im Falle der Funktionsstruktur werden nicht angesprochen. Ob die Flüsse zwischen den Funktionen ebenfalls mit Hilfe von Taxonomien beschrieben werden, oder ob es bei den drei bekannten Größen Energie, Stoff, Information bleibt ist unbekannt. Schließlich wird nicht angegeben, wie es im Entwurfsprozess weiter geht, also was sich an die Funktionsmodellierung anschließt. Die Modellierung von Wirk- bzw. Systemstrukturen wird nicht angesprochen.

3.8 Funktions- und Wirkstrukturverarbeitung beim Konzipieren (Kuttig)

Der Ansatz von Detlef Kuttig beschäftigt sich mit der Modellierung von Funktionen im Rahmen des Systementwurfs [Ku93]. Es werden sowohl Funktionshierarchien als auch Funktionsstrukturen betrachtet, wobei das Hauptaugenmerk auf den Funktionsstrukturen liegt.

3.8.1 Funktionsmodellierung

Die Funktionen der Funktionsstruktur haben festgelegte Ein- und Ausgangsgrößen über die sie verbunden werden. Diese Ein- bzw. Ausgangsgrößen werden jedoch auf einer sehr konkreten, physikalischen Ebene spezifiziert. Es muss beispielsweise genau angegeben werden, wie hoch die Drehzahl ist, oder wie viel elektrische Spannung abgegeben wird. Mit Hilfe von mathematischen Formeln und so genannten Randbedingungsparametern werden aus den Eingangsgrößen die Ausgangsgrößen bestimmt.

Der Ansatz ist im Rahmen des SFB 203 „Rechnerunterstützte Konstruktionsmodelle im Maschinenwesen“ entstanden und wurde mit Hilfe des Softwaresystems KALEIT validiert. Das System KALEIT ermöglicht die Modellierung von Funktionshierarchien und -strukturen.

[16] FMEA = Fehler-Möglichkeit-Einfluss-Analyse

Funktionen werden mit Hilfe der Substantiv-Verb Beschreibungsform beschrieben und erhalten zusätzlich Ein- und Ausgangswerte über die sie verknüpft werden. Bei der Verknüpfung überprüft das System dann, ob diese korrekt war, d.h. ob die Verknüpfung auf Basis der Ein- und Ausgangswerte möglich ist.

Zusätzlich zur Funktionsbeschreibung, beschreibt Kuttig in seiner Arbeit, wie eine Lösungssuche, also die Ermittlung von Systemelementen, prinzipiell aufgebaut sein müsste. Auf Grundlage der Ein- und Ausgangswert müsste die Suche nach Systemelementen erfolgen, die genau diese Ein- und Ausgangswerte besitzen. In der Arbeit bleibt es allerdings bei der Beschreibung wie eine Suche aussehen müsste, eine konkrete Umsetzung mit Hilfe eines Softwaresystems erfolgt nicht.

Das Problem des Ansatzes nach Kuttig ist, das er für die Modellierung von Funktionen im Rahmen des Systementwurfs sehr konkret ist. Um Formeln und genaue Ein- und Ausgangswerte angeben zu können, muss man schon ein sehr genaues Bild von der Lösung bzw. von den einzusetzenden Systemelementen haben.

3.9 Funktionsmodellierung als Grundlage zur Gestaltfindung (Huber)

In der Arbeit von Huber [Hub93] wird ein Konzept zur ganzheitlichen Modellierung maschinenbaulicher Erzeugnisse vorgestellt. Insbesondere die funktionale Produktbeschreibung im Rahmen des Systementwurfs steht dabei im Vordergrund. Huber teilt die funktionale Produktbeschreibung in fünf unterschiedliche Sichten ein:

- die physikalische Sicht
- die formal, abstrakte Sicht
- die zeitlich, funktionsflussbezogene Sicht
- die wirkbezogene Sicht
- die logisch, produkttopologische Sicht.

Im Rahmen der Arbeit werden die Tätigkeiten, die für die Modellierung der einzelnen Sichten notwendig sind, detailliert beschrieben.

3.9.1 Funktionsmodellierung

Von besonderer Bedeutung für diese Arbeit ist die Funktionsmodellierung (formal, abstrakte Sicht). Begonnen wird mit der abstrakten Beschreibung des zu erstellenden Systems mit Hilfe einer Funktionshierarchie. Die Funktionsbeschreibung beruht dabei auf der Substantiv-Verb Beschreibungsform. Ausgehend von einer konkreten Gesamtfunktion (z.B.: „Werkstück handhaben“) wird die Hierarchie so lange untergliedert bis in den Blättern nur noch allgemeine Funktionen (Kombination von Energie, Stoff oder Materie mit speichern, leiten, umformen, wandeln oder verknüpfen) vorhanden sind. Für diese Zerlegung in die allgemeinen Funktionen können regeln definiert werden, die eine automatische Zerlegung ermöglichen.

Basierend auf den allgemeinen Funktionen wird dann die Funktionsstruktur erstellt, d.h. jedes Blatt der Funktionshierarchie wird Bestandteil der Funktionsstruktur und mit Hilfe von Flüssen (Energie, Stoff, Information) verbunden. So ergibt sich ein konkreter Übergang von der Funktionshierarchie zur Funktionsstruktur. Einen ähnlichen Ansatz verfolgt auch Kallmeyer (vgl. 3.3.1).

3.9.2 Wirkstrukturmodellierung

Aufbauend auf der Funktionsstruktur wird dann die Wirkfunktionsstruktur (wirkbezogene Sicht) spezifiziert. Hier werden für jede Funktion der Funktionsstruktur noch der Wirkort und die Wirkrichtung spezifiziert. Gleichzeitig wird aus der Funktionsstruktur auch die Baustruktur (logisch, produkttopologische Sicht) und die funktionale Systemstruktur (logisch, produkttopologische Sicht) abgeleitet. Schließlich lässt sich für jede Funktion ein entsprechender physikalischer Effekt spezifizieren, der die Funktion realisiert (physikalische Sicht). Zur Ermittlung der physikalischen Effekte wurde ein Softwarewerkzeug entwickelt welches die Suche nach physikalischen Effekten unterstützt. Die Suche erfolgt dabei auf Basis von Funktionsmustern, d.h. eine Funktion wird als spezielle Funktion (z.B. „Druck in Kraft wandeln“) gespeichert und auf Basis dessen wird ein physikalischer Effekt ermittelt. Die zur Verfügung stehenden physikalischen Effekte sind in einer entsprechenden Bibliothek hinterlegt.

Die Beschreibung der Funktionen beruht auf der Substantiv-Verb Beschreibungsform, jedoch kann jedes beliebige Substantiv und jedes beliebige Verb verwendet werden. Dadurch wird die Suche nach physikalischen Effekten sehr schwierig. Für jede Funktionsbeschreibung muss daher explizit der zugehörige Effekt angegeben werden. Wurde z.B. ein physikalischer Effekt für die Funktion „Druck in Kraft wandeln“ festgelegt und in der Bibliothek gespeichert, dann wird dieser Effekt nicht gefunden, sobald die Funktion „Druck in Kraft umwandeln“ heißt. Das macht die Suche zum einen sehr ineffizient und zum anderen werden so häufig die gleichen physikalischen Effekte verwendet.

Des Weiteren wird in der Arbeit nicht definiert, wie die verschiedenen Sichten zusammenhängen und wie dieser Zusammenhang abgebildet wird. Dies hat zur Folge, dass bei Änderungen, die während des Systementwurfs häufiger auftreten, die verschiedenen Sichten nicht konsistent zueinander sind und somit u. U. nicht das gleiche mechatronische System modellieren.

3.10 Funktionsmodellierung und Lösungsfindung mechatronischer Produkte (Huang)

Diese Arbeit [Hua01] beschäftigt sich mit der Funktionsstrukturmodellierung, wobei insbesondere der Übergang zwischen der allgemeinen Funktionsstruktur zur speziellen Funktionsstruktur thematisiert wird.

3.10.1 Funktionsmodellierung

Der Übergang von der allgemeinen Funktionsstruktur zur speziellen Funktionsstruktur ist zu komplex als das ihn ein Entwickler problemlos durchführen kann. Während mit der allgemeinen Funktionsstruktur lediglich 30 unterschiedliche Funktionen modelliert werden können sind es bei der speziellen Funktionsstruktur laut Hansen [Han74] 10^5 sein. Diesem Problem begegnet Huang, indem er eine weitere Funktionsstruktur einführt, die als Bindeglied zwischen den beiden zur Anwendung kommt. Diese Funktionsstruktur nennt er kanonische Funktionsstruktur und die dort zum Einsatz kommenden Funktionen werden als kanonische Funktionen bezeichnet. Diese kanonische Funktionen sind aufgeteilt in EWIRKPOTENTIAL, EIMPULS, ESTROM*, ESTROM, EVERSCHIEBUNG, EARBEIT und ELEISTUNG.

In der Arbeit wird dann im Einzelnen vorgestellt wie die kanonische Funktionsstruktur modelliert und wie die Übergänge zwischen den einzelnen Funktionsstrukturen durchzuführen sind.

Die Arbeit beschreibt eine Lösung, um der Komplexität der Funktionsstrukturmodellierung Herr zu werden. Der Einsatz der kanonischen Funktionen ist ein neuer Weg und es muss sich erst zeigen ob sich eine weitere Funktionsstruktur im Rahmen der Funktionsmodellierung durchsetzen wird.

3.11 Modellierung mit der SysML

Die „Systems Modelling Language (SysML)“ [SysML04] wurde von einem Konsortium mehrerer Firmen (z.B.: EADS, Motorola, Artisan, Boing) als Erweiterung der Unified Modeling Language (UML) definiert. Mit ihrer Hilfe soll es möglich sein jegliche Art von Systemen modellieren zu können. Dies können sowohl mechatronische Systeme sein, aber auch Softwaresysteme oder sogar organisatorische Systeme wie Personalstrukturen.

Die SysML basiert auf der UML, Version 2.0 und erweitert diese, indem sowohl vorhandene Diagrammarten angepasst als auch neue Diagrammarten definiert wurden. Es existieren Diagrammarten zur Modellierung von Strukturen, Verhalten, Anforderungen oder von Testfällen.

Eins bietet SysML jedoch nicht: Eine Vorgehensweise, die beschreibt, wie man Systeme mit ihrer Hilfe entwickelt bzw. modelliert. Die SysML ist eine Sammlung von Diagrammarten. Es wird weder ein Vorgehen beschrieben wie die Diagramme zu erstellen sind, noch wie sich die Nutzung der Diagramme in den Entwurfsprozess eingliedert. Dies bleibt, wie auch bei der UML, größtenteils offen.

Bemerkt werden muss an dieser Stelle noch, dass die SysML, zum Zeitpunkt dieser Arbeit, noch kein von der OMG[17] genehmigte, standardisierte Erweiterung der UML ist. Mit einer Standardisierung ist jedoch zu rechnen.

3.11.1 Wirkstruktur-/Systemstrukturmodellierung

Zur Modellierung statischer Strukturen, wie z.B. für die topologische Beschreibung mechatronischer Systeme, stellt die SysML das Assembly-Diagramm zur Verfügung. Das Assembly-Diagramm gleicht der zuvor beschriebenen Wirkstruktur nach Kallmeyer und soll daher an dieser Stelle genauer erläutert werden.

Assembly Diagramm

Das Assembly-Diagramm erweitert die aus der UML stammende „StructuredClasses“ - Bibliothek. Die eigentliche Erweiterung liegt in dem Assembly-Element. Ein Assembly-Element stellt eine abgeschlossene Einheit eines Systems dar (eine Baugruppe im Sinne des Maschinenbaus), wobei diese Einheit selbst wieder aus mehreren Elementen bestehen kann. Elemente innerhalb des Assemblies werden über Ports miteinander verbunden, Abbildung 27 zeigt am Beispiel eines Feder/Dämpfer Systems die Modellierung mit dem Assembly-Diagramm.

[17] Object Management Group – Oberstes Gremium für die Standardisierung der UML

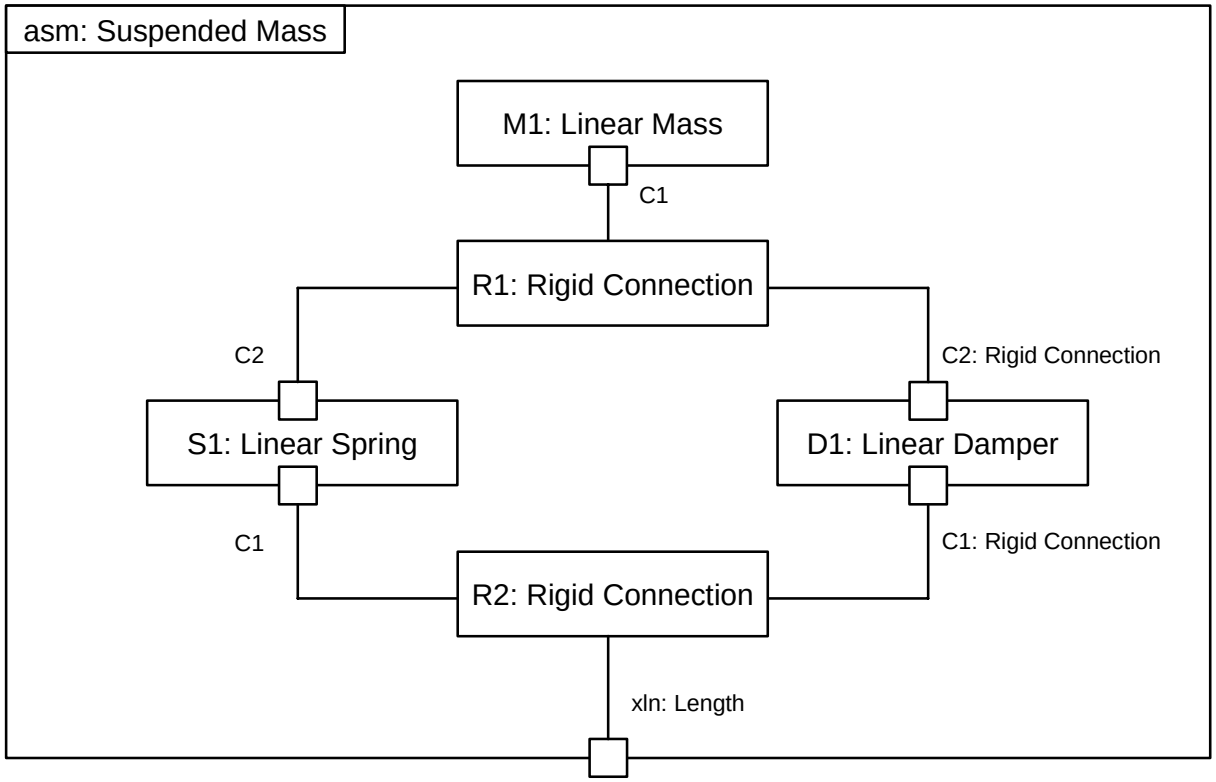


Abbildung 27: Feder/Dämpfer System, modelliert mit der SysML

Die verschiedenen Assemblies können untereinander ebenfalls wieder mit Hilfe von Ports verbunden werden und bilden in ihrer Gesamtheit dann das vollständige System.

Die SysML ist als Erweiterung der UML gedacht und ermöglicht die Modellierung von Systemen jeglicher Art. Insbesondere die Modellierung mechatronischer Systeme ist mit Hilfe des Assembly Diagramms prinzipiell möglich. Das Assembly Diagramm lässt sich mit der Wirkstruktur vergleichen, wie sie von Kallmeyer definiert wurde. Der große Vorteil der SysML liegt in der wesentlich formaleren Beschreibung im Gegensatz zu den bisher vorgestellten Ansätzen.

Die SysML stellt allerdings keine Vorgehensweise zur Verfügung die beschreibt, wann im Entwurfsprozess, sie genau genutzt werden soll. Zusätzlich ist in der SysML keine Diagrammart vorhanden, die die einer Funktionshierarchie bzw. Funktionsstruktur entspricht. Des Weiteren sind keine Konstrukte vorgesehen, die z.B. zwischen einem Hardware- und einem Softwareelement unterscheiden.

Die SysML stellt im Großen und Ganzen nur eine Notation zur Verfügung. Sie ist bisher nicht in den Entwurfsprozess mechatronischer Systeme integriert. Daher gibt es keinen formalen Zusammenhang zu anderen Spezifikationstechniken, wie z.B. der Funktionshierarchie oder Funktionsstruktur.

Abgesehen von den technischen und methodischen Schwächen ist es aus organisatorischer Sicht fraglich, ob sich die SysML durchsetzen wird. In dem Konsortium rund um die SysML fehlen namenhafte Unternehmen die die SysML vorantreiben. Diese namenhaften Unternehmen sind teilweise in anderen Projekten, wie z.B. dem AUTOSAR[18] Projekt vertreten.

[18] AUTOSAR (Automotive Open System Architecture) ist eine Initiative namenhafter Automobilhersteller und Zulieferer bei dem es um die Definition eines Standards für E/E-Anwendungen im Automobil geht. Da über die Inhalte des Projektes keine weiteren Informationen herausgegeben werden, wird es in dieser Arbeit nicht explizit behandelt.

3.12 Schemebuilder

Einen Ansatz zur Beschreibung von Wirkstrukturen liefert das Produkt „Schemebuilder“ der Lancaster University [CPDD].

3.12.1 Wirkstrukturmodellierung

Mit diesem System ist es möglich Strukturen zu modellieren in denen konkrete Elemente wie Controller, Aktoren, oder Positions-Sensoren miteinander verschaltet werden. Die Verschaltung beruht auf den drei Größen Energie, Stoff und Information, wobei diese weiter konkretisiert werden können. Konkretisierung bedeutet, dass jede der drei Größen genauer spezifiziert wird, z.B. kann die Verbindung zwischen zwei Elementen in der Form „Energie.elektrisch.Wechselstrom“ modelliert werden.

Des Weiteren enthält das System eine Bibliothek von Standardkomponenten. Aus der Menge dieser Standardkomponenten lässt sich ein mechatronisches System aufbauen. Die Standardkomponenten sind durch Regeln miteinander Verknüpft wodurch alternative Komponenten ermittelt werden können. Grundlage hierfür ist das regelbasierte System CLIPS. Schließlich ist es möglich für die Komponenten Verhaltensmodelle zu spezifizieren und diese mit Hilfe des Systems Matlab zu simulieren.

Im Schemebuilder wird sehr genau auf die Verschaltung der einzelnen Elemente eingegangen, wodurch eine konkretere Modellierung der Systemstruktur möglich wird. Die Bibliothek von Standardelementen erleichtert die Modellierung. Die Suche nach Alternativen ist ein gutes Hilfsmittel zur Verbesserung der Wirkstruktur.

Nachteil des Schemebuilder ist, dass es nicht in den Entwurfsprozess mechatronischer Systeme eingebunden ist. Wann der Schemebuilder genutzt werden soll ist nicht genau festgelegt. Darüber hinaus sind die Modelle des Schemebuilders nicht mit anderen Modellen, wie z.B. der Funktionshierarchie oder Funktionsstruktur verknüpft. Der Übergang von den Anforderungen zur Lösung in Form einer Systemstruktur wird vom Schemebuilder nur zum Teil unterstützt.

3.13 Schlussfolgerungen und Handlungsbedarf

Der Entwurf mechatronischer Systeme ist ein überwiegend kreativer, auf Wissen und Erfahrung begründetes Vorausdenken technischer Systeme. Dieses Vorausdenken fällt jedoch den Entwicklern zunehmend schwerer, da verstärkt disziplinübergreifendes Wissen erforderlich ist. Wissen und Erfahrungen einzelner Disziplinen reichen alleine nicht mehr aus. So ist es beispielsweise einem Entwickler aus dem Maschinenbau nicht ohne weiteres anzugeben, ob eine vom ihm festgelegte Lösung überhaupt möglich ist, da er Elemente aus dem Bereich der Elektrotechnik nutzten will, die die angenommenen Leistungen gar nicht erfüllen.

Vorgehensweisen, wie z.B. die VDI-Richtlinie 2206, haben sich diesem Problem angenommen und Vorgehen definiert die dabei helfen systematisch mechatronische Systeme zu entwickeln. Großer Wert wird dabei auf die frühe Phase der Entwicklung, den Systementwurf gelegt. Insbesondere die Funktionsmodellierung und die Systemstrukturmodellierung stehen dabei im Vordergrund.

Die natürlichsprachliche Formulierung der Funktionen ist ein gruppendynamischer und langwieriger Prozess, der aufgrund unterschiedlicher Sichtweisen und Fähigkeiten der Entwickler zu enormen Zeit- und Ressourcenaufwendungen führen kann. Es existieren nur wenige Anleitungen und Regeln die die Funktionsmodellierung unterstützen. Diese betreffen:

- die Abstraktion der Gesamtfunktion in Teilfunktionen [VDI97],

- den Hinweis auf Hilfsmittel wie Listen technischer Verben [Bir80],
- Taxonomien der Substantive, Verben und Umsatzarten [Lan00], [Pur03] sowie
- Notationen von Funktionsstrukturen und entsprechende Softwaresysteme [VDI97], [iViP02], [Sch99], [Ku93].

Der Übergang von der Funktionsmodellierung zur Systemstrukturmodellierung ist einer der entscheidenden Schritte im Systementwurf. Hier wird festgelegt welche Disziplin welche Systemelemente im späteren Verlauf der Entwicklungsprozesses herstellen soll. Auch hierfür existieren nur wenige Anleitungen und Regeln. Diese Betreffen

- die Suche nach Wirkprinzipien und Lösungsmuster [PB03] sowie
- die Suche nach physikalischen Effekten [Alt84], [Sch99].

Viele der hier angegebenen bzw. in diesem Kapitel vorgestellten Vorgehensweise und Ansätze sind weitestgehend umgangssprachlich formuliert. Es gibt keine formale Beschreibung weder für die Funktionsmodellierung, noch für die Modellierung von Systemstrukturen. Darüber hinaus gibt es keine formale Beschreibung der Zusammenhänge zwischen der Funktions- und der Systemstrukturmodellierung. Eine Formalisierung ist jedoch erforderlich, wenn eine vollständige und bzgl. den Anforderungen korrekte Prinziplösung modelliert werden soll. Mit Hilfe einer solchen Formalisierung wäre dann eine Rechnerunterstützung möglich, die dabei hilft die Komplexität während des Entwurfs mechatronischer Systeme in den Griff zu bekommen.

Ein entsprechendes Softwaresystem hilft beispielsweise bei der Modellierung der Funktionen und der Systemstruktur. Es ermöglicht die Modellierung der Modelle und stellt sicher, dass diese syntaktisch korrekt sind.

Des Weiteren ist es mit Hilfe einer Rechnerunterstützung möglich für eine gegebene Funktion Systemelemente zu ermitteln, die diese Funktion realisieren können. Dabei können verschiedene Systemelemente und damit verschiedene Lösungen ermittelt werden, wodurch die Kreativität während des Entwurfs gesteigert wird. Der Entwickler wird auf Lösungsalternativen aufmerksam gemacht, an die er vielleicht nicht gedacht hätte.

Schließlich ist es mit einer adäquaten Rechnerunterstützung möglich die Funktionen und die Systemstruktur zueinander konsistent zu halten. Durch Änderungen der Funktionen oder der Systemstruktur kann es beispielsweise dazu kommen, dass am Ende der Modellierung nicht jede Funktion durch ein Systemelement realisiert wird, oder das Systemelemente modelliert wurden, die gar nicht benötigt werden. Die Beziehungen zwischen den Modellen müssen formalisiert werden, indem u.a. festgelegt wird, welche Funktionen durch welche Systemelemente realisiert werden. So lässt sich beispielsweise im Falle einer Löschung nachvollziehen, welche Funktionen oder Systemelemente von dieser Löschung ebenfalls betroffen sind.

Aus diesen geschilderten Situationen ergeben sich folgende Anforderungen an ein Konzept zur rechnerunterstützten Modellierung von Funktionen und Systemstrukturen:

Graphische Modellierung der Funktionen: Zur Modellierung der Funktionen bedarf es einer entsprechenden Notation. Die Modellierung von flussverketteten Funktionsstrukturen soll an dieser Stelle jedoch nicht zum Einsatz kommen, da Funktionsstrukturen bereits Flüsse und damit gewissen Reihenfolgen vorgeben, wodurch bereits eine Lösung impliziert wird. Aus diesem Grund soll die Funktionshierarchie verwendet werden, die in Form einer Baumstruktur dargestellt wird.

Formale Beschreibung der Funktionen: Um u.a. die Suche nach möglichen Lösungen realisieren zu können bedarf es einer entsprechenden Formalisierung der Funktionen. Dies soll auf Grundlage der Substantiv-Verb Beschreibungsform und den von Langlotz [Lan00] und Puri [Pur03] beschriebenen Taxonomien erfolgen.

Suchmechanismus: Auf Basis der formalen Funktionsbeschreibung ist ein Suchmechanismus erforderlich, der nach möglichen Elementen für gegebene Funktionen sucht und damit den Übergang von der Funktionshierarchie zur Systemstruktur beschreibt. Zu diesem Zweck müssen Elemente mit den Funktionen in Beziehung gebracht werden, was eine entsprechende Formalisierung der Elemente erfordert. Für die Suche ist eine entsprechende Menge an Elementen notwendig die zu diesem Zweck in einer Bibliothek hinterlegt werden müssen.

Graphische Modellierung der Systemstruktur: Zur Darstellung der Systemstruktur ist eine entsprechende Notation erforderlich. Diese Notation muss von Entwicklern aller Disziplinen verstanden werden und sollte daher einen eher erklärenden, abstrakten Charakter besitzen. Die Modellierung der Systemstruktur darf nicht zu detailliert sein, da ansonsten das allgemeine Verständnis verloren geht, da zu früh detailliertes Fachwissen erforderlich ist.

Die mit dieser Notation modellierte Systemstruktur soll Systemelemente, wie z.B. Zylinder, Pumpe, oder Regler beinhalten und diese miteinander verbinden können, wodurch eine Struktur (Topologie) des zukünftigen mechatronischen Systems entsteht.

Formale Beschreibung der Systemstruktur: Um die Suche nach Systemelementen und die Konsistenz zwischen Funktionsmodellierung und Systemstruktur gewährleisten zu können (siehe unten), bedarf es einer entsprechenden Formalisierung der Systemstruktur. Darüber hinaus hilft diese Formalisierung dabei, eine modellierte Systemstruktur auf syntaktische Korrektheit hin überprüfen zu können.

Abhängigkeiten: Durch die Nutzung bestimmter Elemente kann es vorkommen, dass dadurch andere Funktionen bzw. andere Elemente zwingend erforderlich sind. Beispielsweise kann die Verwendung einer Pumpe die Konsequenz haben, dass die Funktion „Energie erzeugen“ der Funktionshierarchie hinzugefügt werden muss. Somit entstehen Abhängigkeiten die ein ständiges Wechseln zwischen der Funktionshierarchie und der Systemstruktur zur Folge haben. Diese Abhängigkeiten müssen berücksichtigt und entsprechend modelliert werden können.

Konsistenz: Während des Systementwurfs treten häufig Änderungen sowohl in der Funktionsmodellierung als auch in der Systemstrukturmodellierung auf. Diese Änderungen erfolgen aufgrund von Anforderungsänderungen, oder weil bereits gefällte Entscheidungen rückgängig gemacht werden, wie z.B. durch die Verwendung eines anderen Systemelements für die Realisierung einer Funktion.

Diese ständigen Änderungen müssen überwacht werden, damit am Ende des Systementwurfs jede Funktion durch mindestens ein Systemelement realisiert wird und keine überflüssigen Systemelemente in der Systemstruktur vorhanden sind. Um dies zu gewährleisten sind entsprechende Konsistenzbedingungen zu formulieren und mit Hilfe einer entsprechenden Formalisierung der Beziehungen zwischen den Funktionen und den Systemelementen sicher zu stellen.

KAPITEL 4: BEISPIELHAFTE DARSTELLUNG DES ENTWICKELTEN ANSATZES

Die Analyse des Stands der Technik hat gezeigt, dass es noch keinen Ansatz gibt, der einen rechnerunterstützten Übergang von der Funktionshierarchie zur Systemstruktur ermöglicht. Dieser Übergang, von den lösungsneutralen Funktionen zu den konkreten Elementen des mechatronischen Systems, wird bisher ausschließlich durch den Menschen vorgenommen. Dies fällt jedoch zunehmend schwerer, da durch die Kombination der Elemente der verschiedenen Disziplinen ein riesiger Lösungsraum entsteht, der nur sehr schwer von einzelnen Entwicklern überschaut werden kann.

Zusätzlich wurde festgestellt, dass die Systemstruktur viel zu allgemein ist und deren Spezifikation großes Wissen seitens der Entwickler erfordert. Erst durch die Arbeiten von Kallmeyer ist eine erste, konkrete Modellierungsform hinzugekommen, die es erlaubt, die zu verwendenden Elemente, mit Hilfe einer neuen Notation, in einer Systemstruktur zu modellieren. Diese Systemstruktur zeigt, aus welchen Elementen das mechatronische System besteht und wie diese miteinander in Beziehung stehen.

Am Beispiel der Abstandskontrolle des railcab-Projekts wird im Folgenden, das in dieser Arbeit entwickelte Konzept des rechnerunterstützten Übergangs, zwischen der Funktionshierarchie und der Systemstruktur gezeigt. Hierbei wird aus Gründen des besseren Verständnisses auf Details weitestgehend verzichtet.

4.1 Allgemeine Annahmen

Im Folgenden werden die grundsätzlichen Annahmen, die dieser Arbeit zugrunde liegen, kurz vorgestellt. Hierbei handelt es sich um die Umsetzung der in Kapitel 3.13 aufgeführten Anforderungen an die Modellierung von Funktionshierarchien und Systemstrukturen.

Funktionshierarchie

Der in dieser Arbeit entwickelte Ansatz verwendet auf Seiten der Funktionsmodellierung ausschließlich die Funktionshierarchie. Flussverkettete Funktionsstrukturen werden nicht betrachtet, da sie aufgrund der darin modellierten Reihenfolgen nicht als lösungsneutral angesehen werden. Trotz der Tatsache, dass eine Funktion selbst zwar noch lösungsneutral ist, werden durch die Reihenfolge bereits andere Lösungen ausgeschlossen bzw. impliziert.

Systemstruktur

Die Grundlagen der Systemstrukturmodellierung liefert die Arbeit von Kallmeyer. Insbesondere die von ihm aufgestellte, weitestgehend informale Notation wurde aufgegriffen und gemäß den ermittelten Anforderungen (vgl. Kapitel 3.13) erweitert. Die Komponenten innerhalb der Systemstruktur werden als Systemelemente, oder kurz Elemente, bezeichnet.

Systemelementsuche

Um für eine gegebene Funktion mögliche Systemelemente zu ermitteln, wird ein Suchalgorithmus auf Basis von Graphtransformationsregeln verwendet. Dieser Algorithmus sucht in einer Systemelementbibliothek nach passenden Systemelementen für eine Funktion und schlägt diese dem Entwickler zur Auswahl vor.

Konsistenz der Modelle

Änderungen in einem der Modelle (Funktionshierarchie oder Systemstruktur) können Auswirkungen auf das jeweils andere Modell haben. Diese Auswirkungen müssen explizit betrachtet und behandelt werden, um Inkonsistenzen zwischen den Modellen zu vermeiden. Zu diesem Zweck wurden entsprechende Konsistenzbedingungen aufgestellt und deren Einhaltung mit Hilfe von Graphtransformationsregeln formal spezifiziert.

4.2 Beispiel: Abstandskontrolle

Im Rahmen des Forschungsprojektes „railcab“ der Universität Paderborn (vgl. Kapitel 2.2.2.3) wurde ein schienengebundenes, autonom agierendes Shuttle entwickelt. Dieses Shuttle besteht insgesamt aus drei Modulen, dem Antriebsmodul, dem Spurführungsmodul und dem Feder/Neige-Modul[19]. Mit Hilfe des Antriebsmoduls wurden die in dieser Arbeit entwickelten Ansätze validiert.

Ausgangssituation für ein Beispiel ist der Entwurf einer Abstandskontrolle. Zwei Shuttles sollen in der Lage sein, einen definierten Abstand zwischen ihnen einzuhalten. Da kein Mensch als überwachende Einheit zur Verfügung steht, muss die Abstandskontrolle vollkommen autonom funktionieren.

4.2.1 Anforderungen

Untersuchungen im Rahmen des „railcab“-Projektes haben ergeben, dass durch das Bilden von Konvois erhebliche Energieeinsparungen aufgrund des reduzierten Luftwiderstandes möglich sind. Bereits ab einer Konvoigröße von vier Shuttles lässt sich ökonomischer Fahren, als mit dem ICE oder dem Transrapid [WWW05].

Aufgrund dieser Erkenntnis ist es notwendig, Shuttles möglichst dicht hintereinander fahren zu lassen. Da allerdings kein Mensch in der Lage wäre, die Shuttles entsprechend zu steuern, ist eine zuverlässige und sichere Technologie hierfür erforderlich.

Die Anforderungen besagen, dass ein mechatronisches System in die Shuttle eingebaut werden muss, so dass ein möglichst nahes Auffahren zweier Shuttles möglich ist, ohne dass ein Auffahren und damit ein Unfall erfolgt (vgl. Abbildung 28).

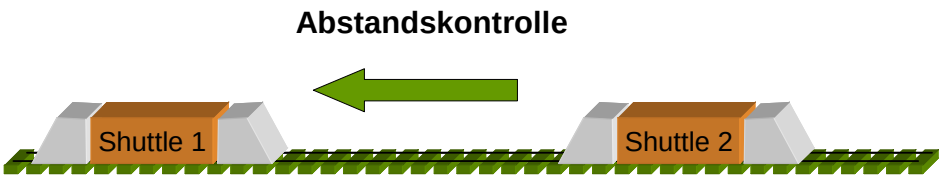


Abbildung 28: Abstandskontrolle

Als Zusatzanforderung wird definiert, dass nur das auffahrende Shuttle (Shuttle 2 in Abbildung 28) die Abstandskontrolle übernimmt. Das vorausfahrende Shuttle (Shuttle 1 in Abbildung 28) soll keinen Einfluss auf die Abstandskontrolle haben. Dies setzt allerdings voraus, dass ein bestimmter Mindestabstand eingehalten werden muss, so dass im Falle einer Vollbremsung der Abstand ausreicht, bis das hintere Shuttle ebenfalls eine Vollbremsung einleiten kann.

Diese Zusatzanforderung schließt aus, dass die beiden Shuttle Daten miteinander austauschen. Dadurch wird es möglich, lediglich ein Shuttle mit seiner Funktionshierarchie und

[19] Ein komplettes Shuttle wird aus zwei identischen Feder/Neige-Modulen zusammengesetzt.

Systemstruktur beschreiben zu müssen und nicht mehrere, was die Komplexität des Beispiels reduziert.

4.2.2 Die ersten Schritte

Nachdem die Anforderungen definiert sind, muss nun eine entsprechende Lösung erarbeitet werden. Zu diesem Zweck wird gemäß VDI-Richtlinie 2206 zunächst die Gesamtfunktion definiert. Für die hier beschriebene Anforderung wurde die Hauptfunktion

„Abstand kontrollieren“

festgelegt. Diese Hauptfunktion ist jedoch viel zu allgemein, als das sofort eine konkrete Lösung festgelegt werden könnte, die alle notwendigen Systemelemente umfasst. Aus diesem Grund wird die Hauptfunktion in weitere Teilfunktionen zerlegt. Diese Zerlegung erfolgt so lange, bis ein gewisser Detaillierungsgrad erreicht ist, d.h. bis es möglich wird Systemelemente anzugeben, die die Teilfunktionen realisieren können.

Abbildung 29 zeigt eine erste Funktionshierarchie. Die Hauptfunktion wurde hier in die Teilfunktionen „Abstand regeln“ und „Ist-Abstand messen“ zerlegt. Da allerdings auch diese Zerlegung noch zu allgemein war, wurden diese Teilfunktionen ebenfalls unterteilt. Dieses Vorgehen wurde immer weiter fortgesetzt. Die gestrichelten Linien deuten an, das die Zerlegung noch nicht beendet ist, sondern hier nur aus Übersichtsgründen weggelassen wurde. Nähere Informationen zu den Funktionen befinden sich in Abschnitt 4.2.3.

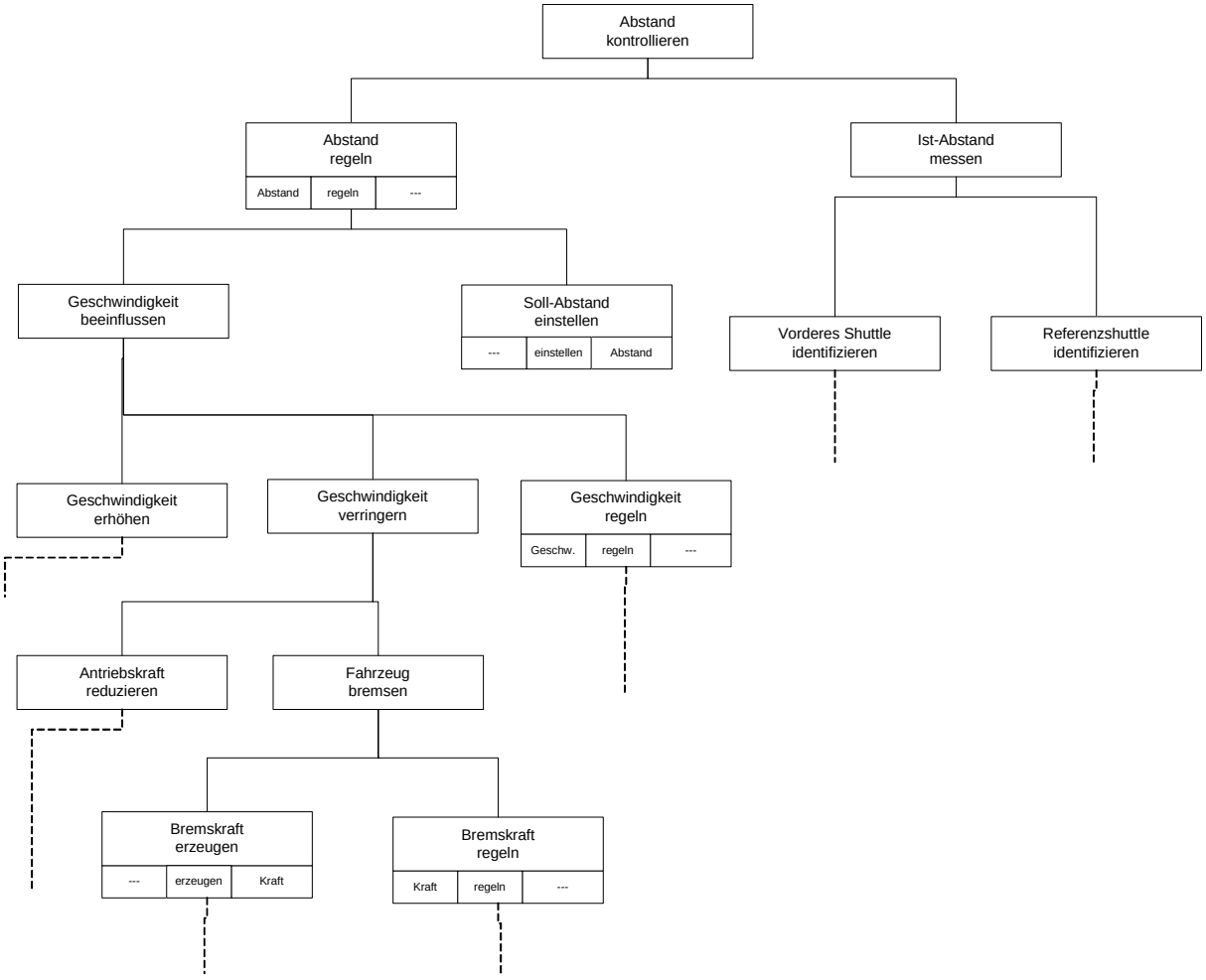


Abbildung 29: Erste Funktionshierarchie

Zerlegungstipp: Es wird vorgeschlagen die Hauptfunktionen nur so lange in Teilfunktionen zu zerlegen, solange gedanklich noch keine Lösung „im Kopf“ vorhanden ist, also solange die Zerlegung noch nicht von einer Lösung beeinflusst wird. Wird die Zerlegung von Systemelementen bereits beeinflusst, dann sollte die Zerlegung gestoppt und zuerst überprüft werden, ob nicht evtl. auch andere Systemelemente als Lösung in Frage kommen könnten. Über diesen Zerlegungstip hinaus gibt es keine Vorschrift, wie die Funktionen zerlegt werden müssen. Es obliegt ausschließlich dem Entwickler, die richtige Zerlegung vorzunehmen.

Nachdem die erste Funktionshierarchie aufgestellt ist und eine weitere Zerlegung nur noch möglich ist, wenn bereits Systemelemente festgelegt wurden, sollten für genau diese Funktionen Systemelemente ermittelt werden. Hierbei kommt ein Algorithmus zum Einsatz, der für eine gegebene Funktion passende Systemelemente ermittelt und diese dem Entwickler zur Auswahl vorschlägt (vgl. 4.2.5). Dadurch lassen sich Vorfixierungen reduzieren und evtl. andere, möglicherweise bessere Lösungskonzepte entwickeln.

4.2.3 Funktionsbeschreibung

Eine automatische Suche nach Systemelementen ist nur dann möglich, wenn eine Beziehung zwischen Systemelementen und den Funktionen in der Funktionshierarchie hergestellt werden kann. Aus diesem Grund ist es u.a. erforderlich, die Beschreibung der Funktionen zu formalisieren.

Eine aus der Literatur bereits bekannte Form zur Beschreibung von Funktionen ist die „Substantiv-Verb“ Beschreibungsform (vgl. Kapitel 3). Diese Art der Beschreibung ist jedoch zu ungenau, da z.B. bei der Funktion „Kraft wandeln“ nicht klar ist in was die Kraft eigentlich gewandelt werden soll. Aus diesem Grund wurde im Rahmen dieser Arbeit die „Substantive-Verb-Substantive“ - Beschreibungsform definiert (vgl. Kapitel 5). Hierbei lassen sich mehrere Substantive als Input, und mehrere Substantive als Output festlegen kombiniert mit einem einzelnen Verb.

Abbildung 30 zeigt die Teilfunktion „Bremskraft erzeugen“ aus der oben dargestellten Funktionshierarchie. Im oberen Teil befindet sich eine umgangssprachliche Beschreibung der Funktion. In den drei Spalten darunter werden die Input-Substantive, das Verb und die Output-Substantive spezifiziert. In der ersten Spalte stehen die Input-Substantive (hier: keine), in der Mitte das Verb (hier: „erzeugen“) und in der rechten Spalte die Output-Substantive (hier: „Kraft“).

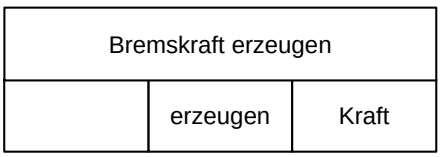


Abbildung 30: Funktion "Bremskraft erzeugen"

Bibliothek von Systemelementen

Neben der formalen Beschreibung der Funktionen ist es für die Suche erforderlich, zum einen eine gewisse Menge an Systemelementen zur Verfügung zu haben und zum anderen eine Beziehung zwischen diesen Systemelementen und der spezifizierten Funktion herstellen zu können.

Um dieses Problem zu lösen, werden Systemelemente innerhalb einer Systemelementbibliothek abgelegt. Dabei werden, neben verschiedenen Attributen des jeweiligen Systemelements, insbesondere deren Realisierungsbeziehungen hinterlegt.

Unter einer Realisierungsbeziehung versteht man die Angabe, welche Funktionen das Systemelement prinzipiell realisieren kann. Dies bedeutet, dass für jedes Systemelement in

der Bibliothek festgelegt wird, welche Funktionen, also welche Substantive-Verb-Substantive Kombinationen es erfüllen kann. Abbildung 31 zeigt beispielhaft ein in der Bibliothek abgelegtes Systemelement („Hydraulikzylinder“) inkl. der von ihm realisierbaren Funktion („Kraft erzeugen“).

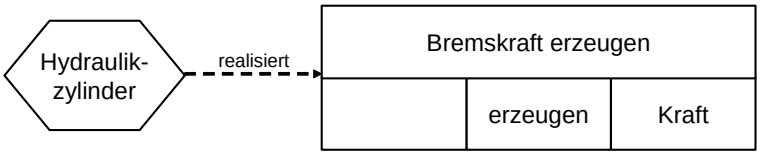


Abbildung 31: In der Bibliothek abgelegtes Systemelement inkl. Realisierungsbeziehung

Durch diese Information ist es nun möglich, die in der Funktionshierarchie beschriebene Funktion mit den Funktionen in der Bibliothek zu vergleichen und so die passenden Systemelemente zu ermitteln.

Hilfsfunktionen

Die Auswahl eines Systemelements kann in einigen Fällen bedeuten, dass noch andere Systemelemente zur Verfügung stehen müssen, damit das gewählte Systemelement seine Aufgabe überhaupt erfüllen kann. So benötigt z.B. ein Radar oder eine Pumpe elektrischen Strom, damit sie funktionieren. Diese Art der Abhängigkeit wird durch Hilfsfunktionen ausgedrückt, wobei sich deren Aufbau nicht von den übrigen Funktionen (Hauptfunktion bzw. Teilfunktionen) unterscheidet.

Beim Speichern der Systemelemente in der Bibliothek werden die von dem jeweiligen Systemelement benötigten Hilfsfunktionen ebenfalls mit gespeichert. So benötigt ein Hydraulikzylinder beispielsweise die Hilfsfunktionen „Volumenstrom regulieren“ und „Volumenstrom erzeugen“.

Nachdem ein Systemelement zur Realisierung einer Funktion ausgewählt wurde, werden die vom ihm benötigten Hilfsfunktionen als Teilfunktionen in die Funktionshierarchie integriert.

Für diese Hilfsfunktionen müssen nun ebenfalls wieder Systemelemente ermittelt werden, wodurch sich ein ständiger Wechsel zwischen der Funktionshierarchie und der Systemstruktur ergibt. Beendet ist dieser Kreislauf erst, wenn keine neuen Hilfsfunktionen mehr hinzukommen.

Eine detaillierte Beschreibung der Funktionen befindet sich in Kapitel 5.

4.2.4 Funktionshierarchie

Abbildung 32 zeigt die Funktionshierarchie aus Abbildung 29, nachdem für einige Funktionen Systemelemente ausgewählt, deren Hilfsfunktionen integriert und die Funktionshierarchie weiter unterteilt wurde. Hilfsfunktionen sind hier gestrichelt dargestellt. Die gewählten Systemelemente sind zum besseren Verständnis oberhalb der jeweiligen Funktionen angegeben.

Der Vorteil durch die Integration der Hilfsfunktionen liegt darin, dass auf diese Weise keine wichtigen Systemelemente vergessen werden können. Würde man die Hilfsfunktionen nicht berücksichtigen, würde unter Umständen erst viel später auffallen, dass noch zusätzliche Systemelemente benötigt werden. Dies könnte dann zu erheblichen Zeit- und Kostenproblemen führen, da aufwändige Änderungen notwendig werden könnten.

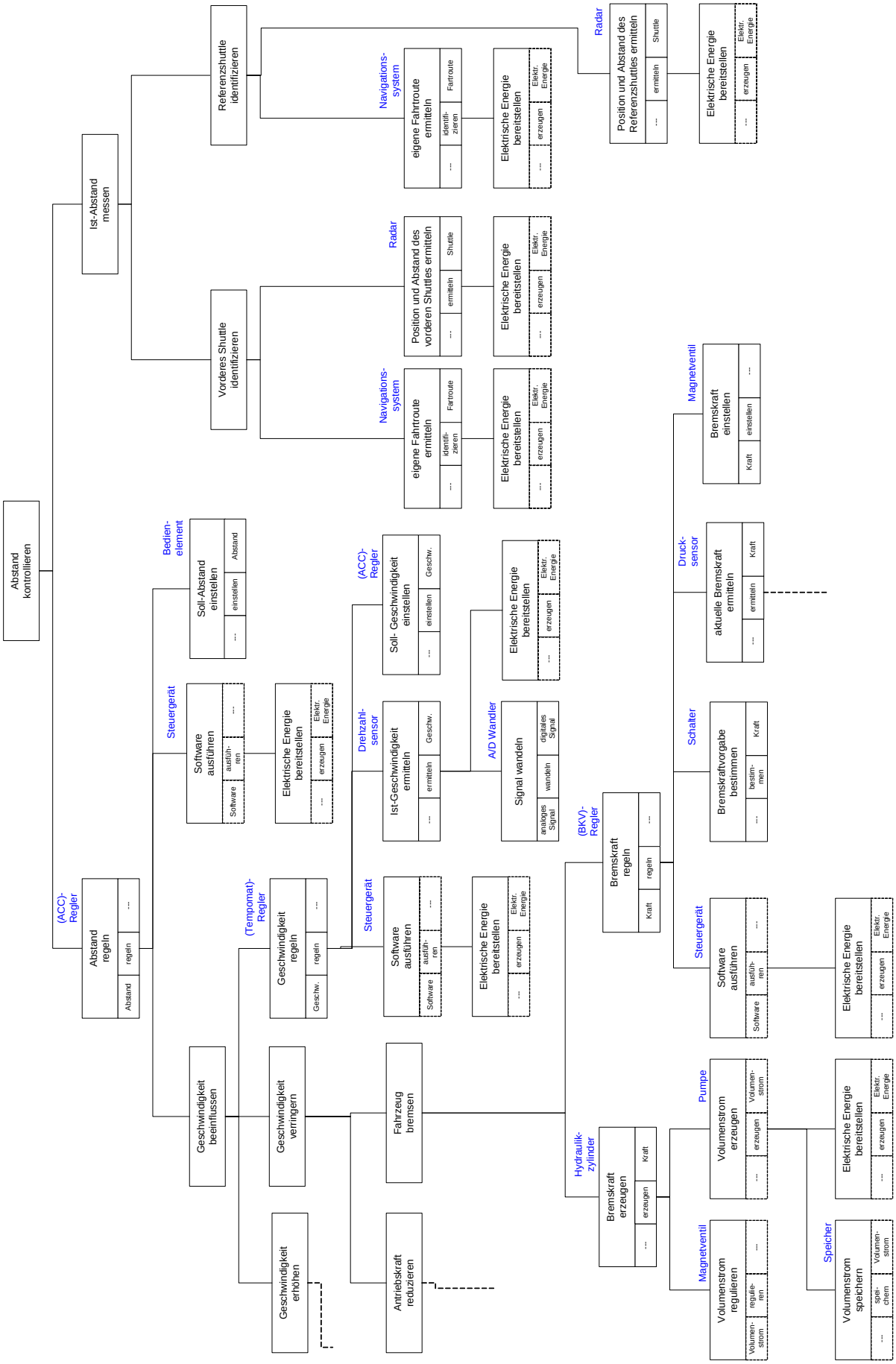


Abbildung 32: Funktionshierarchie der Gesamtfunktion "Abstand kontrollieren"

4.2.5 Suche nach Systemelementen

Nachdem in den vorhergehenden Abschnitten die Funktionshierarchie und die darin spezifizierten Funktionen beschrieben wurden, wird in diesem Abschnitt erläutert, wie für eine einzelne Funktion Systemelemente automatisch ermittelt werden.

Die Suche nach Systemelementen basiert auf der Substantiv-Verb-Substantiv Beschreibungsform. Auf Basis dieser Beschreibungsform wird die Funktion der Funktionshierarchie mit den Funktionen in der Systemelementbibliothek verglichen. Das genaue Vorgehen gliedert sich wie folgt:

1. Im ersten Schritt wird die Systemelementbibliothek durchsucht und ermittelt, welche Systemelemente die gesuchte Funktion erfüllen.
2. Als nächstes werden die gefundenen Systemelemente dem Entwickler präsentiert. Dieser wählt dann das Systemelement aus, das er für richtig hält. Dieses wird dann in die Systemstruktur integriert.
3. Gleichzeitig wird das Systemelement der Funktion zugewiesen. Diese Zuweisung besagt, dass mit Hilfe dieses Systemelements die Funktion realisiert wird.
4. Im Anschluss daran wird überprüft, ob das gewählte Systemelement Hilfsfunktionen besitzt. Ist das der Fall, dann werden diese Hilfsfunktionen als neue Teilfunktionen, in die Funktionshierarchie integriert.
5. Daraufhin wird für die unmittelbar betroffenen Teilfunktionen ein neuer Status berechnet. Insgesamt gibt es drei Stati, „realisiert“, „teilweise realisiert“ und „nicht realisiert“. Diese Stati geben einen Überblick darüber, welche Funktionen bereits realisiert werden und welche noch nicht.
6. Nachdem für alle Funktionen Systemelemente gewählt wurden, stehen diese ohne jegliche Verknüpfungen nebeneinander in der Systemstruktur. Daher muss der Entwickler diese nun zu einer gemeinsamen Systemstruktur verknüpfen.

Der hier skizzierte Algorithmus wird mit Hilfe von UML-Klassendiagrammen und darauf basierenden Graphtransformationsregeln formal beschrieben. Detaillierte Angaben hierzu sind in Kapitel 7 zu finden.

Vorgehen am Beispiel der Abstandskontrolle

Das folgende Beispiel (vgl. Abbildung 33) soll kurz die oben genannte Vorgehensweise verdeutlichen. Zuerst wird nach einem Systemelement für die Funktion „Bremskraft erzeugen“ gesucht. Die Wahl fiel dabei auf einen Hydraulikzylinder. Dieser Hydraulikzylinder benötigt die Hilfsfunktionen „Volumenstrom regulieren“ und „Volumenstrom erzeugen“.

Zur Realisierung der Hilfsfunktion „Volumenstrom regulieren“ wurde ein Magnetventil ausgewählt. Für die Hilfsfunktion „Volumenstrom erzeugen“ wurde eine Pumpe gewählt, wobei diese wiederum die Hilfsfunktionen „Volumenstrom speichern“ und „Elektrische Energie bereitstellen“ benötigt.

Die Hilfsfunktion „Volumenstrom speichern“ wird durch das Systemelement Speicher realisiert und die Hilfsfunktion „Elektrische Energie bereitstellen“ ist noch offen.

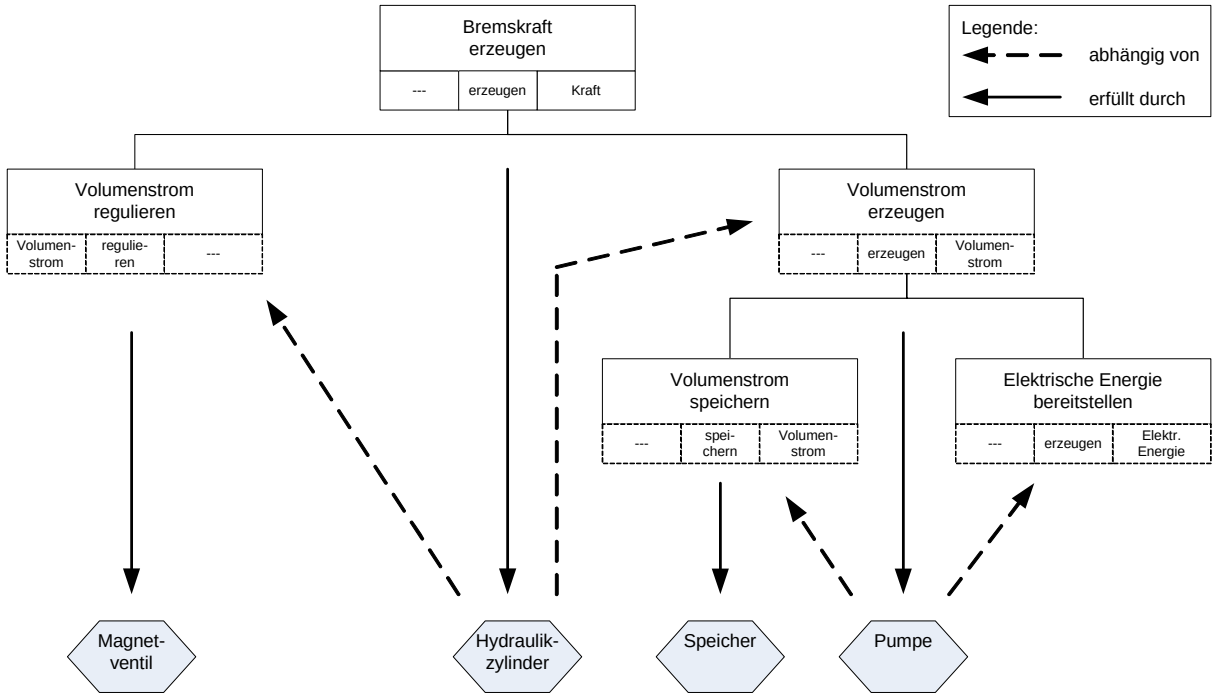


Abbildung 33: Erfüllungsbeziehungen (Ausschnitt)

4.2.6 Systemelemente

Nachdem mit Hilfe der Funktionshierarchie, der Systemelementbibliothek und einem Suchalgorithmus Systemelemente ermittelt wurden, müssen diese nun in einer Systemstruktur miteinander verknüpft werden. Um diese Verknüpfung durchführen zu können, bedarf es einer entsprechenden Darstellungsform. Zu diesem Zweck wurde die von Kallmeyer entwickelte Notation als Grundlage genommen und erweitert.

Jedes Systemelement wird beschrieben durch einen Namen, möglicherweise enthaltenen, internen Systemelementen und deren Verknüpfungen, dem Typ (Hardware oder Software), die Funktionen, die es erfüllt und welche Hilfsfunktionen es benötigt. Zusätzlich lassen sich Attribute spezifizieren, die das Systemelement genauer beschreiben. Eine Systemelementbeschreibung zeigt Abbildung 34.

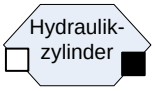
Graphische Darstellung	Beschreibung
	Name: Hydraulikzylinder Typ: Hardware Erfüllende Funktionen: Bremskraft erzeugen (--- | erzeugen | Kraft) Attribute: - max. Kraft: 30 Nm Abhängigkeiten: - Volumenstrom regulieren (Volumenstrom | regulieren | ---) - Volumenstrom erzeugen (--- | erzeugen | Volumenstrom)

Abbildung 34: Beschreibung des Systemelements "Hydraulikzylinder"

Die Verknüpfung der Systemelemente erfolgt mit Hilfe von Ports, die die Schnittstellen nach Außen darstellen. Hierbei gibt es zwei verschiedene Porttypen, einen Input-Port (weißes Quadrat) und einen Output-Port (schwarzes Quadrat). Die Verwendung von Ports zur Beschreibung der Schnittstellen stammt aus der UML 2.0 [UML04] und wurde hier übernommen.

Detaillierte Angaben zu den Systemelementen befinden sich in Kapitel 6.

4.2.7 Systemstruktur

Wie in 4.2.6 bereits gezeigt, werden Systemelemente mit Hilfe von Ports miteinander verknüpft. Mit Hilfe dieser Verknüpfungen können drei verschiedene Arten von Flüssen übertragen werden. Hierzu zählen Energieflüsse, Materialflüsse und Informationsflüsse. Dargestellt werden die Informationsflüsse mit gestrichelten Linien, Stoffflüsse mit dicken, durchgezogenen Linien und Energieflüsse mit dünnen, durchgezogenen Linien. Abbildung 35 zeigt die sich ergebende Systemstruktur der in Abbildung 32 dargestellten Funktionshierarchie.

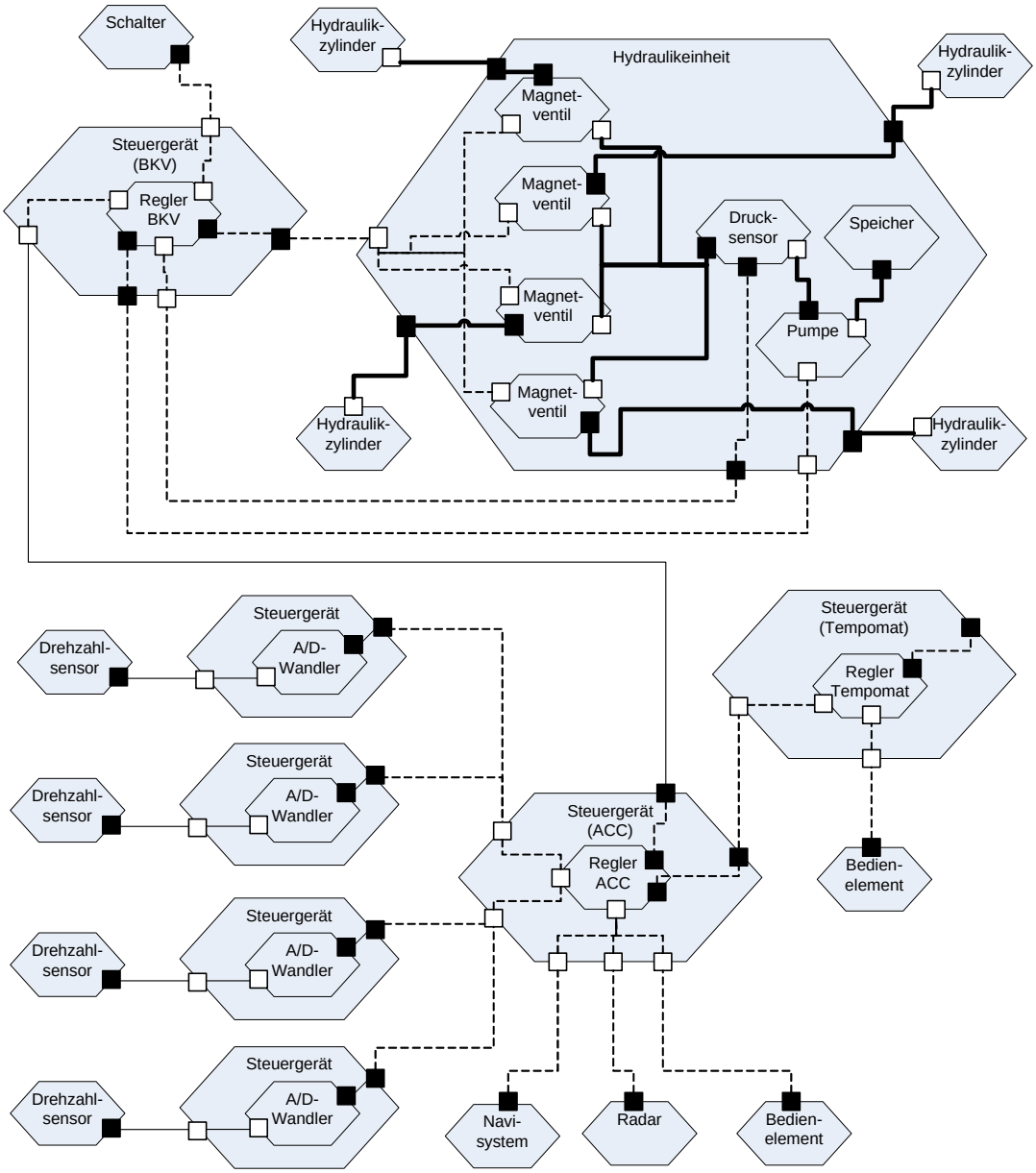


Abbildung 35: Systemstruktur der Abstandskontrolle (Ausschnitt)

Typ / Instanz Problem

Durch die automatische Ermittlung von passenden Systemelementen wird die Modellierung der Systemstruktur deutlich verbessert. Was allerdings mit der Suche nicht gelöst wird, ist die Angabe darüber, wie viele Systemelemente letztendlich in das mechatronische System eingebaut werden sollen. Die Suche liefert lediglich den Typ der Systemelemente, nicht aber ihre genaue Anzahl (Instanzen).

Als Beispiel sei die Funktion „Volumenstrom regulieren“ aus Abbildung 33 gewählt. Die Suche nach passenden Systemelementen liefert das Systemelement „Magnetventil“. Das allerdings genau vier von diesen Magnetventilen in das System eingebaut werden sollen (vgl. Abbildung 35), liefert die Suche nicht. Dies muss der Entwickler auf Basis seiner Erfahrung und seines Wissens entscheiden. Erst durch diese zusätzlichen Informationen, wie z.B. das Wissen über physikalische Gesetze oder Vorstellungen von der räumlichen Anordnung der einzelnen Systemelemente, lässt sich eine Systemstruktur erzeugen.

4.2.8 Konsistenz der Modelle

In den vergangenen Abschnitten wurde die Funktionshierarchie, die Systemstruktur und die automatische Suche nach Systemelementen vorgestellt. Hierbei wurde ein sehr stringentes Vorgehen von der Funktionshierarchie zur Systemstruktur angenommen. Während des Entwurfs ist dieses stringente Vorgehen allerdings in den meisten Fällen nicht einzuhalten. Aufgrund von Anforderungsänderungen und neuen Designentscheidungen kommt es vor, dass Funktionen wieder gelöscht, bereits zugewiesene Systemelemente wieder entfernt oder neue Funktionen hinzugefügt werden müssen.

Diese Anforderungen an die Entwicklung der Funktionshierarchie und der Systemstruktur können zur Folge haben, dass im Laufe der Entwurfszeit beide Modelle immer unübersichtlicher werden und die Zusammenhänge zwischen ihnen nicht mehr nachvollzogen werden können. Wird z.B. ein Systemelement gelöscht muss erkennbar sein, dass die zuvor zugewiesene Funktion nun ein neues Systemelement zu ihrer Realisierung benötigt.

Im Rahmen dieser Arbeit wurden daher Konsistenzbedingungen aufgestellt und formalisiert. Die Formalisierung dieser Konsistenzbedingungen erfolgt mit Hilfe von Graphtransformationsregeln. basierend auf den UML-Klassendiagrammen der Funktionshierarchie und der Systemstruktur.

Die genaue Beschreibung zur Erhaltung der Konsistenz befindet sich in Kapitel 8.

4.3 Zusammenfassung

Der hier beschriebene Ansatz zeigt, wie aus einer lösungsneutralen Funktionshierarchie systematisch eine Systemstruktur erzeugt werden kann (vgl. Abbildung 1). Dabei kommen Systemelemente zum Einsatz, die zur Erfüllung bestimmter Funktionen Lösungen darstellen. Systemelemente bestehen aus einem oder mehreren, miteinander verknüpften Systemelementen und sind in einer Systemelementbibliothek abgelegt.

Durch Abhängigkeiten kommt es vor, dass die Funktionshierarchie dynamisch erweitert wird, d.h. durch die Wahl eines Systemelements kommen immer wieder Funktionen (Hilfsfunktionen) hinzu. Dies führt dazu, dass sich die Funktionshierarchie immer weiter vergrößert, bis schließlich keine neuen Hilfsfunktionen mehr hinzukommen und alle Funktionen durch Systemelemente realisiert werden.

Neben den Änderungen aufgrund von Abhängigkeiten kann es vorkommen, dass sich Anforderungsänderungen oder Designänderungen ergeben und dadurch die Funktionshierarchie und die Systemstruktur angepasst werden müssen. Diese Veränderungen

dürfen jedoch nicht zu Inkonsistenzen zwischen den beiden Modellen führen, d.h. es darf beispielsweise kein Systemelement in der Systemstruktur vorhanden sein, das überhaupt nicht benötigt wird.

Um diesem Problem zu begegnen, wurden im Rahmen dieser Arbeit verschiedene Konsistenzbedingungen aufgestellt und mit Hilfe von Graphtransformationsregeln formal spezifiziert. Dadurch ist sichergestellt, dass nur Modelle erstellt werden, die auch konsistent zueinander sind.

In den folgenden Kapiteln werden die in diesem Kapitel nur grob beschriebenen Ansätze detailliert erläutert. In Kapitel 5 wird die Funktionshierarchie beschrieben und in Kapitel 6 die Systemstruktur. In Kapitel 7 wird die Suche nach den Systemelementen beschrieben, gefolgt von der Beschreibung der Konsistenz in Kapitel 8.

KAPITEL 5: FUNKTIONSMODELLIERUNG

Die Funktionsmodellierung ist ein integraler Bestandteil des Systementwurfs mechatronischer Systeme [vgl. Kapitel 2.2.2.2]. Sie kommt unmittelbar nach der Aufnahme der Anforderungen zum Einsatz und wird dazu verwendet die ausschließlich funktionalen Anforderungen an das zu entwickelnde System lösungsneutral[20] zu strukturieren. Die Funktionsmodellierung dient dann im weiteren Verlauf der Entwicklung anderen Spezifikationen als Vorlage bzw. stellt die Anforderungen für die Spezifikationen bereit.

In der Literatur wird prinzipiell zwischen zwei Modellarten, der *Funktionsstruktur* und der *Funktionshierarchie* unterschieden. Die Vor- und Nachteile dieser Modellarten werden im folgenden Abschnitt diskutiert und begründet, warum im Rahmen dieser Arbeit ausschließlich die Funktionshierarchie zum Einsatz kommt.

Im Anschluss daran wird die Funktionshierarchie detailliert beschrieben, wobei sowohl die Notation, als auch die entsprechende Formalisierung vorgestellt wird. Zum Abschluss des Kapitels wird eine kurze Zusammenfassung gegeben.

5.1 Anforderungen an die Funktionsbeschreibung

In den Kapiteln zwei und drei wurden einige Ansätze vorgestellt, die sich mit der Funktionsmodellierung beschäftigen. Trotz der durchaus unterschiedlichen Ansätze konnten einige Gemeinsamkeiten bzgl. der Anforderungen festgestellt werden. Diese gemeinsamen Anforderungen an die Funktionsmodellierung sollen auch im Rahmen dieser Arbeit zur Anwendung kommen. Darüber hinaus werden noch weitere Anforderungen gestellt, um sowohl die Suche nach Systemelementen, als auch die Konsistenz zur Systemstruktur ermöglichen zu können. Die Anforderungen lauten:

(1) **Verständlichkeit:** Die Funktionsbeschreibungen sollten leicht verständlich sein.

(2) **Lösungsneutralität:** Funktionen sollten weitestgehend Lösungsneutral formuliert werden können.

(3) **Wenig Einschränkungen:** Der bekannte und vertraute Wortschatz eines Entwicklers sollte bei der Funktionsbeschreibung nicht zu stark eingeschränkt werden.

(4) **Automatische Suche:** Funktionen müssen so beschrieben werden, dass eine automatische Suche nach Systemelementen möglich ist.

(5) **Status:** Der Entwickler muss jederzeit in der Lage sein nachzuvollziehen, welche Funktionen bereits durch Systemelemente realisiert werden und welche noch nicht.

Die Erfüllung dieser Anforderungen lässt sich prinzipiell mit Hilfe zweier, unterschiedlicher Ansätze zur Beschreibung von Funktionen erreichen, der Funktionshierarchie oder der Funktionsstruktur.

5.2 Abgrenzung von Funktionshierarchie und Funktionsstruktur

Funktionshierarchien werden im Rahmen des Entwurfs mechatronischer Systeme seit langem eingesetzt [vgl. Kapitel 3.1.1]. Funktionshierarchien beschreiben die funktionalen

[20] Lösungsneutral bedeutet, dass Systemelemente die dem Entwickler bekannt sind, keinen Einfluss auf die Erstellung der Funktionen hat.

Anforderungen an ein Produkt, wobei komplexe[21], auf den ersten Blick unüberschaubare Funktionen, in kleine überschaubare Funktionen zerlegt werden.

5.2.1 Funktionshierarchie

Das Aufstellen der Funktionshierarchie erfolgt typischerweise in zwei Schritten.

(1) Im ersten Schritt werden die komplexen Funktionen in weniger komplexe Funktionen zerlegt, wobei diese Zerlegung lösungsneutral erfolgt, d.h. der Entwickler macht sich noch keine Gedanken darüber, mit welchen physikalischen Effekten oder Systemelementen er die Funktionen später realisieren möchte.

(2) Im zweiten Schritt fließen dann Lösungsideen in die Funktionszerlegung mit ein. Jetzt werden komplexere Funktionen in Funktionen zerlegt vor dem Hintergrund einer konkreten Lösungsidee. Dieser zweite Schritt erfolgt in der Regel, je weiter die Funktionen zerlegt werden, da ab einem bestimmten Punkt eine lösungsneutrale Zerlegung nicht mehr möglich ist.

Als Beispiel sei folgendes Szenario gegeben:

Es soll eine Lenkung für ein Fahrzeug entwickelt werden. Die Funktion die dabei realisiert werden soll ist das leiten einer Kraft (Lenkbewegung) vom Menschen an die Rädern (kurz: „Kraft leiten“). Diese Funktion ist lösungsneutral, da noch keine konkreten Vorstellungen einer Lösungsidee vorhanden ist (vgl. obigen Schritt 1).

Als nächstes wird nun die lösungsneutrale Funktion „Kraft leiten“ in weitere Funktionen zerlegt, die ihrerseits eine Lösungsidee erkennen lassen. Die Zerlegung erfolgt in die Funktionen „Kraft in Moment wandeln“, „Moment in Kraft wandeln“, „Moment leiten“ und „Kraft leiten“ (vgl. Abbildung 36).

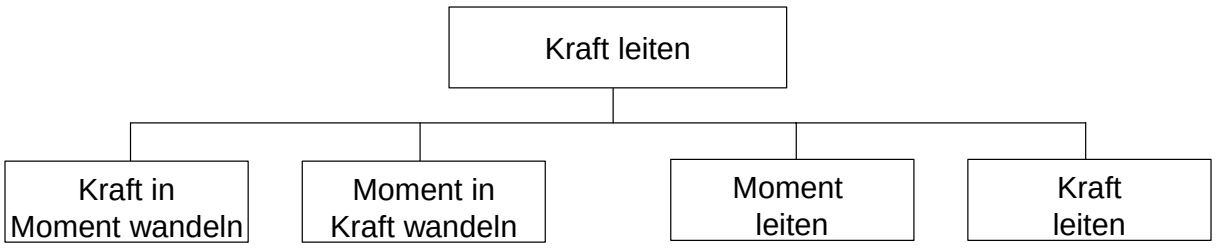


Abbildung 36: Beispiel einer Funktionshierarchie

Die Lösungsidee, die sich hinter dieser Zerlegung verbirgt, ist die klassische Lenkung eines Kraftfahrzeugs. Daher kann diese Zerlegung auch nicht mehr als lösungsneutral angesehen werden (vgl. obigen Schritt 2). Zuerst wird die Kraft des Menschen in ein Moment gewandelt (Lenkrad), dann wird dieses Moment geleitet (Lenkwelle), dann in eine Kraft gewandelt (Zahnrad) und diese Kraft an die Räder geleitet (Spurstange).

Diese Lösung stellt natürlich nur eine von vielen Lösungen dar. So könnte man z.B. auch die vom Menschen erzeugte Kraft direkt über ein Lenkgestänge an die Räder leiten, ohne vorher in ein Moment zu wandeln. Eine andere Lösung wäre die Wandlung der Kraft in eine Information, die Übertragung dieser Information, die Wandlung dieser Information in eine Kraft und die Übertragung dieser Kraft auf die Räder (Steer-by-Wire).

Die Tatsache das es mehrere Zerlegungsmöglichkeiten und damit mehrere Lösungsmöglichkeiten gibt zeigt, dass es nicht genau eine Funktionshierarchie gibt, sondern

[21] Unter Komplexität wird in diesem Zusammenhang der Grad der Übersichtlichkeit des Zusammenhangs zwischen Eingang und Ausgang, die Vielschichtigkeit der notwendigen physikalischen Vorgänge sowie die sich ergebende Anzahl der zu erwartenden Baugruppen und Einzelteile verstanden. In Anlehnung an [Pur03]

mehrere geben kann. Diese unterschiedlichen Lösungsideen müssen allerdings vom Entwickler erkannt und in Form einer Funktionshierarchie entsprechend umgesetzt werden.

5.2.2 Funktionsstruktur

An dieser Stelle der Funktionszerlegung setzt als nächstes die Funktionsstruktur an. Die aus der Funktionshierarchie stammenden Funktionen werden zuerst um Ein- und Ausgangswerte ergänzt. Dann werden die untersten Funktionen der Funktionshierarchie (Blätter in der Funktionshierarchie) miteinander verknüpft. Die Verknüpfung erfolgt dabei mit Hilfe von Energie-, Stoff- oder Informationsflüssen (vgl. Kapitel 3.1.1 und 3.3.1). Bezogen auf das obige Beispiel ergibt sich folgende Funktionsstruktur (vgl. Abbildung 37).

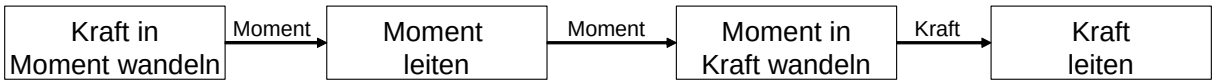


Abbildung 37: Beispiel einer Funktionsstruktur

5.2.3 Vergleich von Funktionshierarchie und Funktionsstruktur

An dieser Stelle lassen sich nun folgende Beobachtungen in Bezug auf die Modellierung der Funktionshierarchie und Funktionsstruktur machen:

(1) Die Zerlegung der lösungsneutralen Funktionen innerhalb der Funktionshierarchie ist nur möglich, wenn bereits konkrete Lösungsideen (in Form von Systemelementen) vorhanden sind. Viel Erfahrung, Wissen und Kreativität sind dabei notwendig, um aus einer Menge von Systemelementen gedanklich die richtigen auszuwählen.

(2) Zusätzlich zu Punkt 1 kommt hinzu, dass durch die Wahl der Systemelemente nicht nur die Funktionserfüllung erreicht wird, sondern meistens auch gleich deren Verknüpfungen fest stehen. Lösungen entstehen immer durch die Kombination von Systemelementen und selten durch die Auswahl eines einzelnen Elements.

Bezüglich dieser Punkte stellt sich die Frage, welchen zusätzlichen Nutzen die Modellierung der Funktionsstruktur bringt. Wenn der Entwickler bereits in konkreten Systemelementen denkt, dann liegt es nahe auch gleich die Systemelemente miteinander zu verknüpfen, ohne den Umweg über die Funktionsstruktur zu gehen.

Aufgrund der Tatsache, dass die Modellierung der Funktionsstruktur bzgl. der beobachteten Punkte und der aufgestellten Anforderungen (vgl. 5.1) keinen entscheidenden Mehrwert bringt, wird in dieser Arbeit ausschließlich die Funktionshierarchie zur Modellierung von Funktionen verwendet.

Im Folgenden wird die Funktionshierarchie im Detail vorgestellt, wobei insbesondere auf die Änderungen eingegangen wird die zur Umsetzung oben beschriebener Anforderungen (vgl. 5.1) erforderlich sind.

5.3 Aufbau der Funktionshierarchie

Die Funktionshierarchie wird in Form einer Baumstruktur dargestellt (vgl. Abbildung 38). In der Wurzel des Baumes steht die *Gesamtfunktion*. Diese wird dann in weitere Funktionen (*Teilfunktionen*) zerlegt.

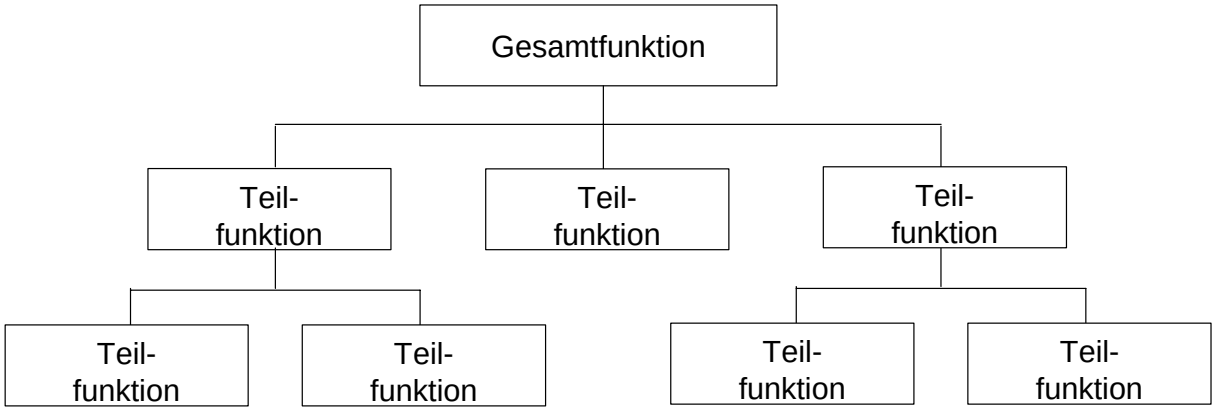


Abbildung 38: Baumstruktur der Funktionshierarchie

Nach der Zerlegung der Gesamtfunktion in Teilfunktionen wird versucht Systemelemente für die entstandenen Teilfunktionen zu finden. Gelingt dies nicht, wird die Funktionshierarchie weiter zerlegt. Dieser Prozess setzt sich solange fort, bis die Teilfunktionen so überschaubar sind, dass entsprechende Systemelemente zuzuordnen sind.

Eines der Ziele dieser Arbeit ist es, basierend auf den Funktionen der Funktionshierarchie automatisch diejenigen Systemelemente zu ermitteln, die in der Lage sind die Funktionen zu realisieren (vgl. Anforderung (4) aus Abschnitt 5.1). Zu diesem Zweck müssen die Funktionen auf bestimmte Art und Weise beschrieben werden. Wie diese Beschreibung aussieht, zeigt der folgenden Abschnitt.

5.4 Funktionsbeschreibung

Zur Beschreibung von Funktionen sind aus der Literatur (vgl. Kapitel 3.1 - 3.3) verschiedene Möglichkeiten bekannt. Drei der verbreitesten Beschreibungsformen sind:

(1) eine textuelle Beschreibung.

(2) eine strukturierte Beschreibung in Form eines Substantivs und eines Verbs (Substantiv-Verb Beschreibungsform).

(3) eine strukturierte Form mit Ein- und Ausgangswerten und einer Zustandsänderung (Black-Box Prinzip).

Entscheidend ist an dieser Stelle zu überprüfen, welche dieser Beschreibungsformen geeignet ist, um eine automatische Suche nach Systemelementen zu ermöglichen.

Bewertung der Beschreibungsformen

Die Beschreibung von Funktionen mit Hilfe frei gewählter Texte (Punkt 1) ist für eine automatische Suche nach Systemelementen nicht sinnvoll. Durch eine beliebige Beschreibung der Funktion ist eine gezielte Suche annähernd unmöglich.

Die Beschreibung von Funktionen mit Hilfe der Substantiv-Verb Beschreibung (Punkt 2) ist zwar strukturiert und damit prinzipiell für eine Suche zu gebrauchen, jedoch ist sie in einigen Fällen unzureichend. Als Beispiel sei die Funktion „Kraft wandeln“ genannt. Hier wäre es hilfreich angeben zu können, in was die Kraft gewandelt werden soll (z.B. in ein Moment).

Hieraus folgt, dass nur das Black-Box Prinzip in Frage kommt, also eine Funktionsbeschreibung mit Eingangswerten, Ausgangswerten und einer entsprechenden Transformation.

5.4.1 Black-Box Prinzip zur Beschreibung von Funktionen

Der Aufbau einer Funktion nach dem „Black-Box“ Prinzip gliedert sich in vier Bereiche,

- einen beschreibenden Text,
- die Eingangswerte,
- die Transformationsgröße und die
- Ausgangswerte.

Der beschreibende Text ist frei wählbar und dient ausschließlich dem besseren Verständnis der Funktion. Die Eingangswerte werden der Funktion übergeben, mit Hilfe der Transformationsgröße bearbeitet und in Form der Ausgangswerte wieder abgegeben.

Mit Hilfe des beschreibenden Textes wird die unter 5.1 festgelegte Anforderung (1), „Funktionsbeschreibungen sollten leicht verständlich sein“, erfüllt.

Zur Beschreibung der Eingangs- und Ausgangswerte und der Transformationsgröße wird auf die bekannte Substantiv-Verb Beschreibungsform zurückgegriffen. Die Eingangs- bzw. Ausgangswerte werden mit Hilfe von Substantiven und die Transformationsgröße durch Verben beschrieben.

- Das *Substantiv* benennt den Objektbereich, mit oder an dem bzw. durch den etwas geschieht oder geschehen soll [Pur03, S. 47].
- Das *Verb* gibt an, was in dem vom Substantiv angegebenen Bereich oder durch diesen geschieht bzw. geschehen soll [Pur03, S. 47].

Durch die Zerlegung der Funktionen und der Beschreibung mit Hilfe von Substantiven und Verben wird die unter 5.1 festgelegte Anforderung (2), „Funktionen sollten weitestgehend Lösungsneutral formuliert werden können.“ erfüllt.

Diese Form zur Beschreibung von Funktionen heißt „Black-Box“ Prinzip, weil offen ist, WIE die Funktion realisiert wird, also wie die Transformation durchgeführt wird.

5.4.2 Notation

Wie in Abbildung 39 dargestellt, wird eine Funktion als Rechteck mit vier Bereichen dargestellt. Im oberen Teil steht der frei formulierbare Text der Funktion. Auf der linken Seite stehen die Eingangswerte (Substantive), in der Mitte die Transformationsgröße (Verb) und auf der rechten Seite die Ausgangswerte (Substantive).

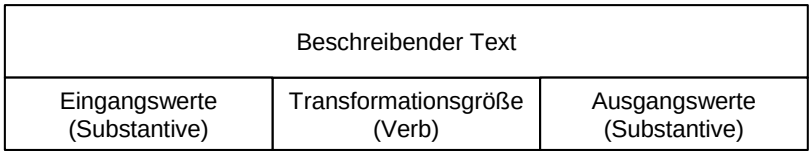


Abbildung 39: Funktionsaufbau

Abbildung 40 zeigt die Funktion „Elektrisch unterstütztes Wandeln einer Kraft in ein Moment“, bei der die beiden Eingangswerte „Kraft“ und „Elektrischer Strom“ in ein „Moment“ gewandelt werden.

Elektrisch unterstütztes Wandeln einer Kraft in ein Moment		
Kraft Elek. Strom	wandeln	Moment

Abbildung 40: Beispiel einer Funktion

Nachdem gezeigt wurde, wie Funktionen beschrieben werden, wird im folgenden Abschnitt erläutert, wie diese Beschreibung bei der Suche nach Systemelementen genutzt wird.

5.5 Wahl der Substantive und Verben

In Abschnitt 5.4.1 wurde gezeigt, dass sich eine Funktion aus Substantiven (Eingangs-/Ausgangswerten) und einem Verb (Transformationsgröße) zusammensetzt. Diese Beschreibungsform soll dabei helfen, gemäß Anforderung (4) aus 5.1, eine automatische Suche nach Systemelementen zu ermöglichen. Basierend auf den Eingangs- und Ausgangswerten und der Transformationsgröße soll ermittelt werden, welche Systemelemente in der Lage sind, diese Transformation zu erfüllen (vgl. Kapitel 7).

Entscheidend für die Suche nach Systemelementen sind die Substantive und die Verben. Durch die Wahl von ein oder mehreren Substantiven und einem Verb entstehen Kombinationen. Diese Kombinationen werden dazu genutzt, um Systemelemente zu finden, die diese Kombinationen erfüllen können. Hierbei ist darauf zu achten, dass die Substantive und Verben nicht willkürlich gewählt werden können, sondern die Menge aus der sie gewählt werden im Vorfeld festgelegt wird. Nur so wird ein Vergleich zwischen einer gesuchten Kombination und einer von einem Systemelement erfüllten Kombination möglich.

Aus der Literatur sind zwei Ansätze bekannt, die sich mit der Festlegung von Substantive und Verben zur Beschreibung von Funktionen beschäftigen. Hierbei handelt es sich zum einen um die *allgemeinen Funktionen* (vgl. Kapitel 3.1.1) und zum anderen um die *speziellen Funktionen* (vgl. Kapitel 3.2.1). Im Folgenden werden diese Ansätze vorgestellt auf ihre Brauchbarkeit hin analysiert.

5.5.1 Allgemeine Funktionen

Allgemeine Funktionen setzen sich aus den drei allgemeinen Größen *Energie, Stoff* und *Information* (Eingangs-/Ausgangswerte) und den fünf Zustandsänderungen *Speichern, Leiten, Umformen, Wandeln* und *Verknüpfen* (Transformationsgrößen) zusammen. Sie wurden von Roth [Roth94] definiert und in Form der VDI-Richtlinie 2222 [VDI97] detailliert [vgl. Kapitel 3.1].

Allgemeine Funktionen sind nach der Substantiv-Verb Beschreibungsform aufgebaut, z.B. „Energie wandeln“ oder „Stoff leiten“. Die Struktur lässt sich aber ohne Probleme erweitern um Ein- und Ausgangswerte gleichermaßen berücksichtigen zu können. Beispiele für solche Erweiterungen sind:

- Energie -> leiten -> Energie (z.B. Lenksäule die ein Drehmoment leitet)
- Stoff -> wandeln -> Energie (z.B. Verbrennungsmotor der eine Kraft erzeugt)
- Energie, Information -> verknüpfen -> Energie (z.B. ein Lichtschalter)

Mit Hilfe der allgemeinen Funktionen werden Probleme wie Mehrdeutigkeiten und unterschiedliche Abstraktionsebenen vermieden, da sie nur aus einer kleinen Menge an Substantiven (drei) und Verben (fünf) bestehen. Diese Einschränkung widerspricht jedoch der Anforderung (3) aus Abschnitt 5.1.

Ein zusätzliches Problem der allgemeinen Funktionen lässt sich anhand der drei gezeigten Beispiele erkennen Es ist sehr schwer sich bei den Beispielen etwas Konkretes vorzustellen. Erst durch eine genauere Beschreibung der Funktion (siehe Klammern hinter den Funktionen) wird klar, was damit eigentlich gemeint ist. Ein Entwickler muss starke Fähigkeiten zur Abstraktion besitzen, um mit so wenig Substantiven und Verben auszukommen.

Aufgrund der Tatsache, dass die Anforderung (3) nicht erfüllt ist und darüber hinaus das Problem bzgl. der Abstraktionsfähigkeit auftritt, kommt man zu dem Schluss, dass die allgemeinen Funktionen ungeeignet sind. Aus diesem Grund wurden u.a. von Pahl/Beitz [PB03] oder Langlotz [Lan00] spezielle Funktionen eingeführt.

5.5.2 Spezielle Funktionen

Die speziellen Funktionen erweitern die allgemeinen Funktionen, indem sie die Anzahl der zu verwendenden Substantive und Verben nicht mehr einschränken. Es kann jedes Substantiv und jedes Verb bei der Beschreibung der Funktion verwendet werden.

Ursprünglich werden spezielle Funktionen auch in der Substantiv-Verb Beschreibungsform spezifiziert. Diese lässt sich aber, wie auch bei den allgemeinen Funktionen, dahingehend erweitern, dass sowohl Ein- als auch Ausgangswerte berücksichtigt werden.

Die in 5.5.1 beschriebenen Beispiele würden als spezielle Funktionen wie folgt spezifiziert:

- Drehmoment -> übertragen -> Drehmoment
- Benzin -> umwandeln -> Kraft
- Strom, Impuls -> schalten -> Strom

Anhand dieser Funktionsbeschreibung lässt sich wesentlich genauer beschreiben, was eigentlich erreicht werden soll und ermöglicht dem Entwickler die Substantive und Verben zu nutzen, die ihm geläufig sind.

Durch die Verwendung beliebiger Substantive und Verben wird die unter 5.1 festgelegte Anforderung (3), „Der bekannte und vertraute Wortschatz eines Entwicklers sollte bei der Funktionsbeschreibung nicht zu stark eingeschränkt werden.“ erfüllt.

Bei der Beschreibung mit beliebigen Substantiven und Verben treten allerdings folgende Probleme auf:

Mehrdeutigkeit: Aufgrund der Vielfalt der verfügbaren Worte (Substantive oder Verben) kann es vorkommen, dass einige Worte unterschiedlich geschrieben, aber von ihrer Bedeutung her gleich sind (z.B.: „wandeln“ und „umwandeln“).

Unterschiedliche Konkretisierungsgrade: Aufgrund der Vielfalt der verfügbaren Worte kann es vorkommen, dass die verwendeten Worte unterschiedliche Detaillierungsgrade aufweisen (z.B.: „Flüssigkeit“ und „Benzin“).

Aufgrund dieser Probleme kann es vorkommen, dass bei der Suche nach Systemelementen in einem bestimmten Fall (z.B.: Verb „wandeln“) ein passendes Systemelement gefunden wird, in einem anderen Fall (z.B.: Verb „umwandeln“) jedoch nicht.

Auf der einen Seite gibt es also die Anforderung, dass es eine ausreichende Menge an Substantiven und Verben geben muss (Anforderung 3 aus 5.1), auf der anderen Seite hat diese Menge dann zur Folge, dass aufgrund der Mehrdeutigkeiten und unterschiedlichen Konkretisierungsgrade die Suchergebnisse falsch bzw. ungenau sein können.

Nach der Analyse der speziellen Funktionen kommt man zu dem Schluss, dass diese in ihrer ursprünglichen Form demnach nicht ausreichen, um Funktionen zu beschreiben und gleichzeitig die Suche nach Systemelementen zu ermöglichen. Im Rahmen dieser Arbeit wurde daher ein Ansatz gewählt, der die speziellen Funktionen als Ausgangspunkt hat und dahingehend erweitert, dass die oben erwähnte Anforderung erfüllt und gleichzeitig die angesprochenen Probleme nicht auftreten. Wie diese Erweiterungen aussehen, zeigt der folgende Abschnitt.

5.6 Beziehungen innerhalb der Substantive und innerhalb der Verben

Ausgangspunkte bei der in Kapitel 7 beschriebenen Suche nach Systemelementen sind die Substantive und Verben einer Funktion. Um die Suche technisch durchführen zu können, müssen die zur Verfügung stehenden Substantive und Verben vorgegeben werden. Aus dieser vorgegebenen Menge kann der Entwickler dann auswählen und die Funktion entsprechend beschreiben.

Zur Erfüllung der Anforderung (3) aus Abschnitt 5.1, muss die Menge der Substantive und Verben entsprechend groß sein. Dies führt jedoch zu den oben bereits erwähnten Problemen der Mehrdeutigkeit und unterschiedlichen Konkretisierungen.

Um dies Problem zu lösen, müssen die Substantive und Verben untereinander in Beziehung gesetzt werden. Mit Hilfe von *Äquivalenz-* und *Konkretisierungsbeziehungen* ist es möglich eine ausreichend große Menge an Substantive und Verben zur Verfügung zu stellen, ohne das die Suchergebnisse falsch oder unvollständig sind. Besteht z.B. eine Äquivalenzbeziehung zwischen den Verben „wandeln“ und „umwandeln“, so werden bei einer Suche in beiden Fällen die gleichen Systemelemente ermittelt.

Im Folgenden werden die beiden Beziehungsarten im Detail vorgestellt.

5.6.1 Äquivalenzbeziehungen

Um semantisch gleichbedeutende Substantive und Verben berücksichtigen zu können, muss für jedes Substantiv und für jedes Verb angegeben werden, zu welchem anderen Substantiv bzw. zu welchem anderen Verb es semantisch äquivalent ist[22]. Eine solche Beziehung wird im Rahmen dieser Arbeit als Äquivalenzbeziehung bezeichnet.

Äquivalente Substantivgruppen: Substantive die durch eine Äquivalenzbeziehung mit anderen Substantiven verknüpft sind bilden Gruppen äquivalenter Substantive (vgl. Abbildung 41). Jedes Substantiv gehört zu genau einer äquivalenten Gruppe, auch wenn es das einzige Substantiv dieser Gruppe ist.

Äquivalente Verbgruppen: Verben die durch eine Äquivalenzbeziehung mit anderen Verben verknüpft sind bilden Gruppen äquivalenter Verben (vgl. Abbildung 41). Jedes Verb gehört zu genau einer äquivalenten Gruppe, auch wenn es das einzige Verb dieser Gruppe ist.

[22] Eine semantische Äquivalenz zwischen Substantiven und Verben ist nicht zulässig. Substantive können nur zu anderen Substantiven semantisch äquivalent sein. Gleiches gilt für die Verben.

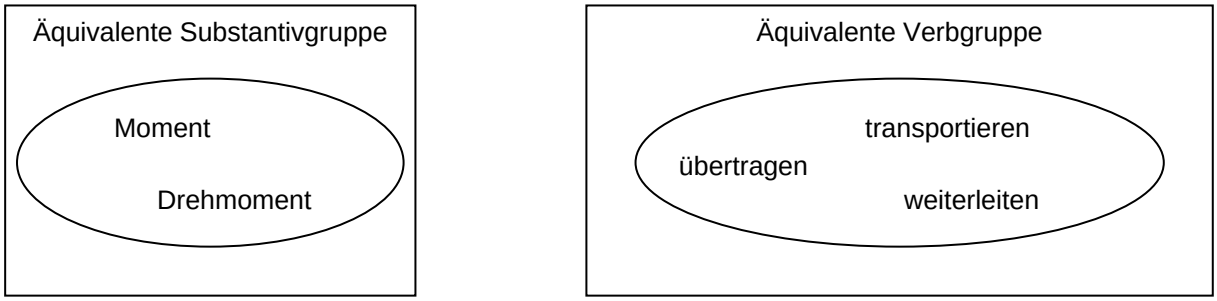


Abbildung 41: Beispiele für äquivalente Substantiv- bzw. Verbgruppen

Wie sich die äquivalenten Substantiv- und Verbgruppen auf die Suche auswirken, wird in Kapitel 7.6.4 erläutert.

Aufstellen der äquivalenten Gruppen

Die Wahl der richtigen Äquivalenzbeziehungen ist ein gruppendynamischer Prozess, in dem sich alle Entwickler auf einen gemeinsamen Nenner einigen. In diesem Fall bedeutet dies, dass im Vorfeld einer Produktentwicklung durch das Entwicklerteam festgelegt werden muss, welche Substantive und Verben prinzipiell verwendet werden und wie die Äquivalenzbeziehungen zwischen diesen aussehen.

5.6.2 Unterschiedliche Konkretisierungsgrade

Zuzüglich zu den Äquivalenzbeziehungen kommt es vor, dass Substantive und Verben unterschiedliche Konkretisierungsgrade aufweisen. Als Beispiel seien die Verben „ändern“ und „vergrößern“, oder die Substantive „Flüssigkeit“ und „Wasser“ genannt.

Durch das Festlegen einer Konkretisierungsbeziehung lassen sich Systemelemente für eine allgemein beschriebene Funktion ermitteln, die ursprünglich für eine wesentlich konkreter beschriebene Funktion vorgesehen war. Wird beispielsweise nach der Funktion „Flüssigkeit - befördern -Flüssigkeit“ gesucht, dann wird bei der Suche eine „Ölpumpe“ gefunden, obwohl diese mit der Funktion „Öl - leiten - Öl“ in der Systemelementbibliothek abgelegt wurde.

Wie an diesem Beispiel zu erkennen ist, stehen die Substantive und Verben hier in einer hierarchischen Beziehung zueinander. Einige Substantive und Verben sind abstrakt und werden von anderen Substantiven oder Verben konkretisiert. Um eine Vergleichbarkeit zwischen abstrakten und konkreten Substantiven und Verben zu ermöglichen und damit die Suche nach Systemelementen zu unterstützen, ist es notwendig diese Form der Beziehung zu spezifizieren.

Die Spezifizierung der Konkretisierungsbeziehung erfolgt auf Basis der in Abschnitt 5.6.1 beschriebenen Gruppen äquivalenter Substantive und Verben. Eine Konkretisierungsbeziehung wird nicht direkt zwischen den Substantive und Verben, sondern zwischen den äquivalenter Gruppen spezifiziert.

Konkrete Substantivgruppen: Eine äquivalente Substantivgruppe die Substantive einer anderen äquivalenten Substantivgruppe konkretisiert wird als konkretere Substantivgruppe bezeichnet (vgl. Abbildung 42).

Konkrete Verbgruppen: Eine äquivalente Verbgruppe die Verben einer anderen äquivalenten Verbgruppe konkretisiert wird als konkretere Verbgruppe bezeichnet (vgl. Abbildung 42).

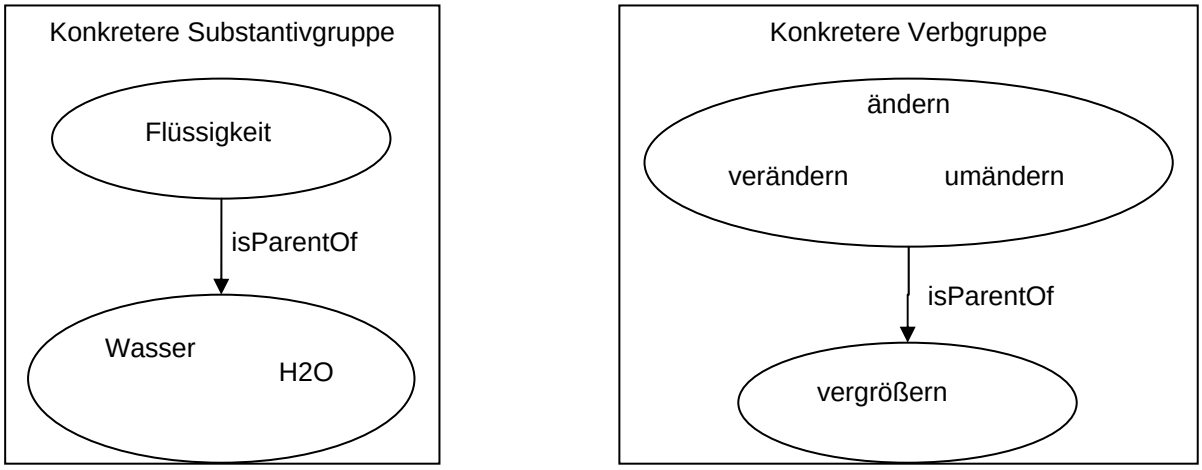


Abbildung 42: Konkrete Substantiv- und Verbgruppen

In Abbildung 42 stellt die jeweils obere äquivalente Gruppe die abstraktere Gruppe dar und die untere Gruppe die konkretere, was durch die Beziehung „isParentOf“ ausgedrückt wird.

Wie die Konkretisierungsbeziehungen bei der Suche berücksichtigt werden, wird in Kapitel 7.6.3 erläutert.

Aufstellen der Konkretisierungsbeziehungen

Ähnlich wie bei dem Aufstellen der äquivalenten Gruppen müssen auch die Konkretisierungsbeziehungen von einem Entwicklerteam festgelegt werden.

5.6.3 Zusammenfassung

Mit Hilfe der beiden Beziehungsarten (Äquivalenzbeziehung und Konkretisierungsbeziehung) ist es möglich eine Menge von Substantiven und Verben zu definieren die bei der Modellierung der Funktionen genutzt und zur Ermittlung passender Systemelemente verwendet werden können.

Durch das Festlegen einer bestimmten Menge an Substantiven und Verben unter Verwendung von Äquivalenz- und Konkretisierungsbeziehungen wird die unter 5.1 festgelegte Anforderung (4), „Funktionen müssen so beschrieben werden, dass eine automatische Suche nach Systemelementen möglich ist.“ erfüllt.

Durch die flexible Menge an Substantiven und Verben wird die Anforderung (3), „Der bekannte und vertraute Wortschatz eines Entwicklers sollte bei der Funktionsbeschreibung nicht zu stark eingeschränkt werden“, erfüllt.

Die konkreten Auswirkungen der beiden unterschiedlichen Beziehungen bei der Suche nach Systemelementen, werden in Kapitel 7.6 erläutert.

5.7 Hilfsfunktionen

Auf Basis der Funktionsbeschreibung lässt sich eine automatische Suche nach Systemelementen durchführen. Bei dieser Suche (vgl. Kapitel 7) werden passende Systemelemente ermittelt und dem Entwickler zur Auswahl vorgeschlagen. Dieser wählt dann eins der vorgeschlagenen Systemelemente aus und integriert dieses in die Systemstruktur. Durch diese Wahl kann es allerdings vorkommen, dass neue Funktionen zur Funktionshierarchie hinzukommen. Dies liegt daran, dass einige Systemelemente von anderen Systemelementen bzw. von anderen Funktionen abhängig sind. So funktioniert z.B. eine

Pumpe nur dann, wenn elektrische Energie zur Verfügung steht. Die so hinzukommenden Funktionen werden *Hilfsfunktionen* genannt.

Hilfsfunktionen sind Funktionen die nicht unmittelbar zur Erfüllung der Gesamtfunktion beitragen. Sie entstehen, wenn zur Erfüllung einer Funktion (Gesamt-, Teil- oder Hilfsfunktion) ein oder mehrere Systemelemente gewählt wurden, die ihrerseits eine oder mehrere andere Funktionen benötigt. Hilfsfunktionen kommen also während des weiteren Entwurfs hinzu, wenn für die Teilfunktionen Systemelemente zugewiesen werden [PB04, S. 43].

Hilfsfunktionen werden in die Funktionshierarchie integriert und stellen dort wieder Teilfunktionen dar. Um sie während des Entwurfs unterscheiden zu können, besitzen sie ein etwas anderes graphisches Erscheinungsbild als die sonstigen Teilfunktionen.

5.7.1 Notation

Das graphische Erscheinungsbild von Hilfefunktionen, die Notation (vgl. Abbildung 43), orientiert sich an der Notation der Gesamt- bzw. Teilfunktionen. Der einzige Unterschied besteht darin, dass die Funktion mit Hilfe einer gestrichelten Linie dargestellt wird.

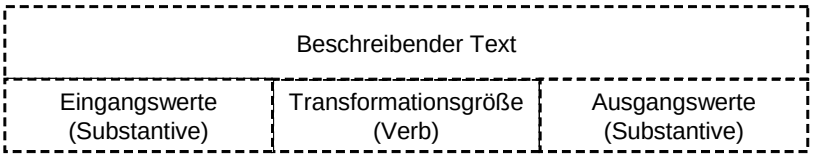


Abbildung 43: Notation der Hilfsfunktionen

Von welchem Systemelement die Hilfsfunktion stammt, wird nicht explizit angegeben. Dies ist im entsprechenden Softwaremodell festgehalten (vgl. Kapitel 8).

5.8 Status einer Funktion

Jede Funktion muss am Ende des Systementwurfs durch mindestens ein Systemelement realisiert werden. Aufgrund der Komplexität und der Abhängigkeiten (vgl. 5.7) kann jedoch die Funktionshierarchie sehr groß werden. Dadurch kann es für einen Entwickler schwer nachzuvollziehen sein, für welche Funktionen bereits Systemelemente ausgewählt wurden und für welche noch nicht.

Um diesem Problem zu begegnen werden drei verschiedene Stati für die Funktionen vergeben. Jeder Funktion wird genau ein Status zugewiesen und zeigt dem Entwickler an, welche Funktionen bereits realisiert werden, welche erst teilweise und welche noch gar nicht.

Jede Funktion hat immer einen der folgenden Stati:

- nicht realisiert
- teilweise realisiert
- realisiert

Wie und wann diese Stati vergeben werden, wird in Kapitel 8 ausführlich erläutert, daher seinen sie an dieser Stelle, nur kurz erwähnt.

Durch die Vergabe eines Status wird die unter 5.1 festgelegte Anforderung (5), „Der Entwickler muss jederzeit in der Lage sein nachzuvollziehen, welche Funktionen bereits durch Systemelemente realisiert werden und welche noch nicht.“ erfüllt.

5.9 Korrekte Funktionsbeschreibung

Nachdem in den vorhergehenden Abschnitten gezeigt wurde, wie eine Funktion beschrieben wird und welche Einschränkungen und Ergänzungen dabei berücksichtigt werden müssen, stellt sich noch die Frage, wann eine Funktion *korrekt* beschrieben ist.

Zur Beschreibung einer korrekten Funktion müssen vier Regeln eingehalten werden:

(1) Jede Funktion hat genau eine Transformationsgröße (Verb)

(2) Jede Funktion hat 0 bis n viele Eingangswerte (Substantive)

(3) Jede Funktion hat 0 bis n viele Ausgangswerte (Substantive)

(4) Jede Funktion hat mindestens einen Eingangs- oder einen Ausgangswert

Diese Regeln wurde festgelegt, um die bekannte Substantiv-Verb Beschreibungsform beibehalten zu können. Des Weiteren ist es für den in Kapitel 7 beschriebenen Algorithmus notwendig, dass für eine Funktion immer ein Verb und ein Eingangs- oder Ausgangswert angegeben ist.

Die Überprüfung, ob die Regeln eingehalten werden, ist nicht Bestandteil dieser Arbeit. Aus diesem Grund gibt es keinen Formalismus, der die Regeln überprüft.

Abgesehen von den bisher nur informal beschriebenen Angaben zur Modellierung von Funktionen ist eine entsprechende Formalisierung ebenfalls vorzunehmen. Nur mit Hilfe einer Formalisierung lässt sich ein entsprechender Suchalgorithmus entwickeln (vgl. Kapitel 7), oder die Konsistenz zwischen der Funktionshierarchie und der Systemstruktur sicherstellen (vgl. Kapitel 8).

5.10 Formalisierung

Dieser Abschnitt zeigt die Formalisierung der Funktionshierarchie mit Hilfe eines UML-Klassendiagramms. Jede Klasse des Diagramms wird beschrieben und der Bezug zu den vorhergehenden Abschnitten hergestellt.

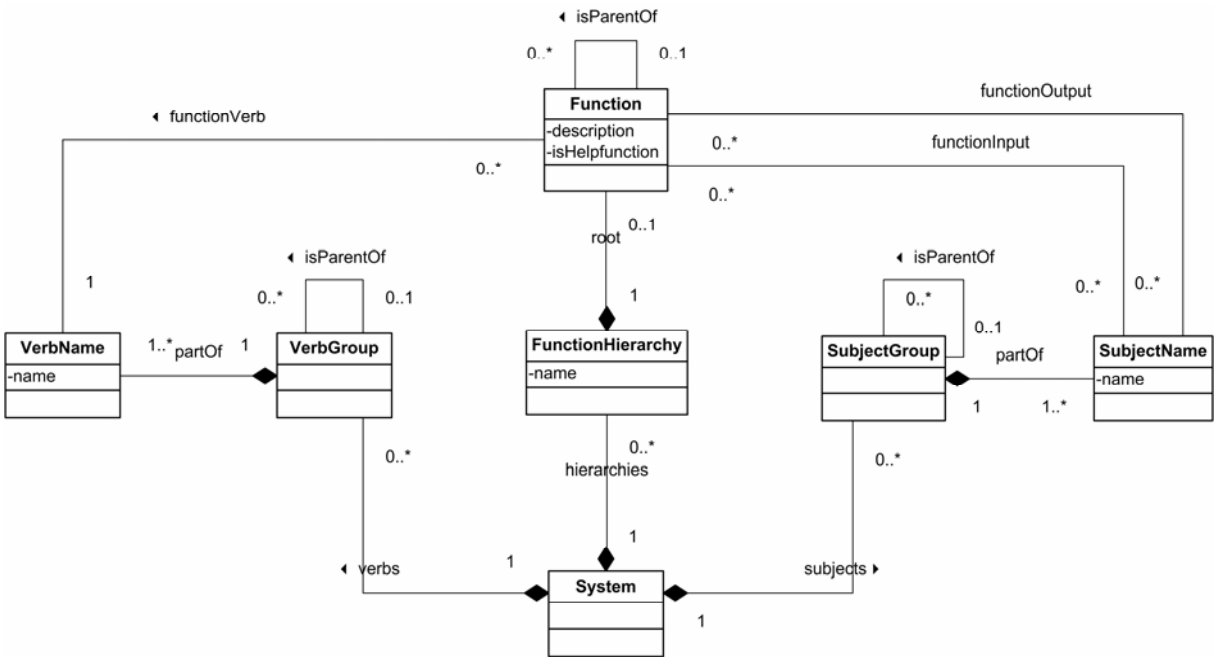


Abbildung 44: Klassendiagramm zur Beschreibung der Funktionen

Kurzbeschreibung der Klassen:

System: Diese Klasse modelliert das Gesamtsystem, also die Menge aller Funktionshierarchien. Alle grundlegenden Konfigurationen wie z.B. die Menge aller Substantive (Assoziation subjects) und die Menge aller Verben (Assoziation verbs) sind hier hinterlegt. Über die Assoziation hierarchies werden alle vorhandenen Funktionshierarchien erreicht.

FunctionHierarchy: Diese Klasse modelliert eine einzelne Funktionshierarchie. Durch die Assoziation root lässt sich der Wurzelknoten, also die Gesamtfunktion adressieren.

VerbGroup: Diese Klasse modelliert eine Gruppe von äquivalenten Verben. Jeder Gruppe ist durch die Assoziation partOf mindestens ein Name (Klasse: VerbName) zugewiesen. Mit Hilfe der Assoziation isParentOf lassen sich die Verben erreichen, die eine Konkretisierung bzw. Verallgemeinerung der Gruppe der Verben darstellen.

VerbName: Diese Klasse modelliert genau ein konkretes Verb. Jedes Verb gehört zu einer Gruppe äquivalenter Verben (Klasse Verb bzw. Assoziation partOf). Einer Funktion wird immer genau ein Verb (Assoziation functionVerb) zugewiesen (vgl. Regel (1) aus 5.8).

SubjectGroup: Diese Klasse modelliert eine Gruppe von äquivalenten Substantiven. Jeder Gruppe ist durch die Assoziation partOf mindestens ein Name (Klasse SubjectName) zugewiesen. Mit Hilfe der Assoziation isParentOf lassen sich die Substantive erreichen, die eine Konkretisierung bzw. Verallgemeinerung der Gruppe von Substantiven darstellen.

SubjectName: Diese Klasse modelliert genau ein konkretes Substantiv. Jedes Substantiv gehört zu einer Gruppe äquivalenter Substantive (Klasse Subject bzw. Assoziation partOf). Einer Funktion werden über die Assoziationen functionInput und functionOutput die jeweils ein- und ausgehenden Substantive zugewiesen.

Function: Diese Klasse modelliert die eigentliche Funktion. Sie verbindet die ein- und ausgehenden Substantive mit einem Verb. Durch die Assoziation isParentOf lassen sich die Teilfunktionen (Kinderknoten) bzw. übergeordneten Funktionen (Väterknoten) erreichen. Mit Hilfe des Attributs isHelpfunction lässt sich spezifizieren, dass die Funktion eine Hilfsfunktion ist.

5.11 Zusammenfassung

Das vorliegende Kapitel beschreibt die Funktionsmodellierung, die im Rahmen dieser Arbeit entwickelt und verwendet wird. Zu Beginn werden Anforderungen an die Funktionsmodellierung aufgestellt und mit bereits bekannten Arten der Funktionsmodellierung (Funktionsstruktur und Funktionshierarchie) verglichen. Anhand dessen wird gezeigt, warum die Funktionshierarchie als Beschreibungsmittel ausgewählt wurde. Im Anschluss daran wird gezeigt, in welcher Form die einzelnen Funktionen der Funktionshierarchie beschrieben werden müssen, um eine automatische Suche nach Systemelementen zu ermöglichen. Die Grundlage der Funktionsbeschreibung liefert die aus der Literatur bereits bekannte *spezielle Funktion.* Diese wurde um einige Aspekte erweitert, um allen aufgestellten Anforderungen gerecht zu werden. Abschließend wird noch auf Hilfsfunktionen eingegangen und definiert, wie eine korrekt beschriebene Funktion aussieht. Abschließend wird anhand eines UML-Klassendiagramms die Struktur der Funktionshierarchie formalisiert.

KAPITEL 6: SYSTEMSTRUKTUR

Im letzten Kapitel wurde die Modellierung der Funktionshierarchie detailliert vorgestellt. Basierend auf der Funktionshierarchie müssen nun Systemelemente bestimmt werden, die diese Funktionen realisieren. Dies erfolgt mit Hilfe der Systemstruktur, die im Folgenden vorgestellt wird.

Eine Systemstruktur setzt sich aus miteinander verknüpften Systemelementen (kurz: Elementen) zusammen. Sie beschreibt den prinzipiellen Aufbau eines mechatronischen Systems, also welche Elemente vorhanden sind und wie sie miteinander verbunden werden. Das Verhalten oder die räumliche Anordnung der Elemente wird mit der Systemstruktur nicht modelliert.

In Kapitel 3 wurden gezeigt, dass sowohl eine graphische als auch formale Beschreibung der Systemstruktur notwendig ist. Die ersten Arbeiten auf diesem Gebiet wurden von Ferdinand Kallmeyer [Kall98] (vgl. Kapitel 3.3) durchgeführt, indem er mechatronische Systeme analysierte und eine neue Notation definierte, mit denen sie modelliert werden können.

Aufgrund ihres informellen Charakters reicht die Arbeit von Kallmeyer allerdings nicht aus. Aus diesem Grund wurde sie im Rahmen dieser Arbeit Formalisiert. Zusätzlich zur Formalisierung wurde die Systemstrukturmodellierung konzeptionell erweitert, um sie an einigen Stellen klarer und ausdrucksstärker zu machen. Die Formalisierung wurde mit Hilfe der Unified Modelling Language (UML) durchgeführt.

Im Folgenden wird nun die Systemstruktur im Einzelnen erläutert. Zuerst werden die Anforderungen an die Systemstruktur spezifiziert. Im Anschluss folgt die Beschreibung der einzelnen Bestandteile der Systemstruktur und wie sie die Anforderungen im Einzelfall erfüllen.

6.1 Anforderungen an die Systemstrukturmodellierung

Ähnlich wie an die Funktionshierarchie werden diverse Anforderungen auch an die Systemstruktur gestellt. Folgenden Anforderungen, die insbesondere im Rahmen des SFB 614 entwickelt wurden, werden an sie gestellt:

(1) **Graphische Modellierung:** Die Systemstruktur muss graphisch modelliert werden können und sollte für Entwickler der verschiedenen Disziplinen leicht verständlich sein.

(2) **Verknüpfungen:** Systemelemente müssen mit Hilfe unterschiedlicher Flussarten miteinander verknüpft werden können.

(3) **Eigenschaften:** Die Angabe von Merkmalen bzw. Eigenschaften für jedes Systemelement muss möglich sein.

(4) **Realisierungsbeziehungen:** Es muss möglich sein festzulegen, dass ein Systemelement in der Lage ist Funktionen zu realisieren.

(5) **Hilfsfunktionen** Mögliche Abhängigkeiten von Hilfsfunktionen müssen berücksichtigt werden.

(6) **Hierarchische Systemelementen:** Die Modellierung von hierarchischen Systemelementen muss möglich sein.

(7) **Strukturierung:** Sowohl Hardware-, Software- als auch logische Elemente müssen gemeinsam modelliert werden können.

6.2 Systemstruktur

Die Grundlage zur Beschreibung der Systemstruktur liefert die Arbeit von Kallmeyer [Kal98]. In dieser Arbeit werden einige graphische Konstrukte beschrieben und deren Einsatz bzw. Bedeutung umgangssprachlich festgelegt (vgl. Kapitel 3.3). Da die Beschreibung jedoch nur informal vorliegt und die oben geschilderten Anforderungen weitestgehend nicht erfüllt werden, musste eine entsprechende Erweiterung vorgenommen werden.

Grundsätzlich besteht eine Systemstruktur aus miteinander verknüpften Systemelementen (vgl. Abbildung 35 in Kapitel 4), wobei die Verknüpfung mit Hilfe unterschiedlicher Flüsse erfolgt. Der genaue Aufbau der Systemelemente wird im Folgenden vorgestellt.

6.3 Systemelemente

Systemelemente innerhalb der Systemstruktur repräsentieren sowohl Elemente aus der Disziplin des Maschinenbaus (z.B. Zylinder, Ventil oder Pumpe), aus dem Bereich der Elektrotechnik (z.B. Steuergerät) oder aus dem Bereich der Informationstechnik (z.B. Regler).

Darüber hinaus werden Systemelemente in drei Klassen eingeteilt. In die Klasse der Hardwareelemente, Softwareelemente und in die Klasse der logischen Elemente. Letztere gruppieren typischerweise die anderen beiden Klassen (z.B. MFM aus Kapitel 2.1.2).

Jedes Systemelement hat unterschiedliche Fähigkeiten, wodurch es sich von anderen Systemelementen unterscheidet. Diese Fähigkeiten drücken sich darin aus, indem ein Systemelement in der Lage ist verschiedene *Funktionen* zu realisieren bzw. zu deren Realisierung beizutragen.

Zusätzlich zu den Funktionen gibt es noch weitere Eigenschaften die ein Systemelement kennzeichnen. Hierzu zählen beispielsweise Gewicht, Nennspannung, Messbereich usw. Diese Eigenschaften lassen sich in Form von *Attributen* für jedes Systemelement festlegen.

Die Nutzung von Systemelementen bringt es mit sich, dass zusätzliche Anforderungen entstehen, d.h. durch die Verwendung eines Systemelements kommen neue, Funktionen (Hilfsfunktionen) zur Funktionshierarchie hinzu. Dies bedeutet das zusätzliche Systemelemente erforderlich sind, die die neu hinzugekommenen Funktionen erfüllen. Es bestehen also *Abhängigkeiten* zwischen Systemelementen und Funktionen.

Um eine Systemstruktur modellieren zu können, müssen die Systemelemente miteinander verbunden werden. Zu diesem Zweck werden *Ports* verwendet die entsprechende Schnittstellen darstellen.

Aufgrund der Komplexität der Systemstruktur ist es erforderlich diese zu verringern, um den Überblick zu behalten und um sich auf das wesentliche konzentrieren zu können. Zu diesem Zweck können *hierarchische Systemelemente* modelliert werden. Hierarchische Systemelemente bestehen selbst wieder aus anderen Systemelementen. Ein Systemelement, das sich in einem anderen Systemelement befindet wird als *internes Systemelement* bezeichnet. Im Gegensatz dazu wird ein nicht hierarchisches, bzw. nicht internes Systemelement als *atomares Systemelement* bezeichnet.

Neben diesen unterschiedlichen Angaben zu einem Systemelement, die in den folgenden Abschnitten im Einzelnen erläutert werden, können die Systemelemente und damit die Systemstruktur graphisch dargestellt werden.

6.3.1 Notation

Die graphische Darstellung von Systemelementen (vgl. Abbildung 45) stammt aus der Arbeit von Kallmeyer [Kal98, S. 93]. Ein Systemelement wird als Sechseck mit dem Namen in der Mitte dargestellt.

Abbildung 45: Graphische Darstellung eines Systemelements

6.3.2 Formalisierung der Systemstruktur

Die Formalisierung der einzelnen Bestandteile der Systemstruktur wird mit Hilfe von UML-Klassendiagrammen vorgenommen. Das folgende Klassendiagramm (vgl. Abbildung 46) zeigt einen Ausschnitt aus dem Meta-Modell zur Beschreibung der kompletten Systemstruktur. Die hier aufgeführten Klassen beschreiben die Systemstruktur und die zugehörigen Systemelemente. Vertiefende Aspekte werden in den anschließenden Abschnitten beschrieben und ergänzt.

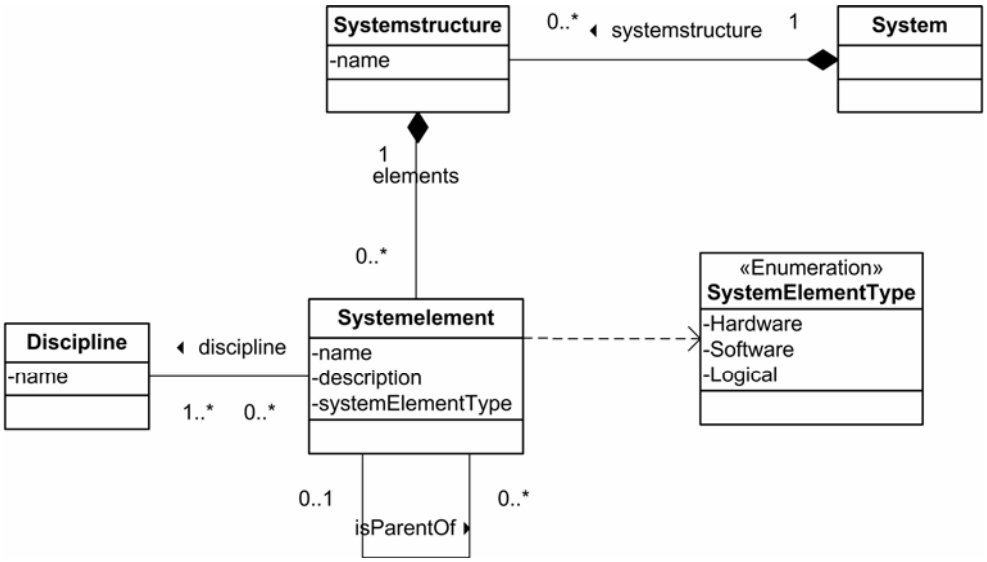


Abbildung 46: Meta-Modell der Systemstruktur (Ausschnitt)

Kurzbeschreibung der Klassen:

System**:** Diese Klasse modelliert das Gesamtsystem, also die Menge aller Systemstrukturen.

Systemstructure**:** Diese Klasse modelliert eine Systemstruktur. Sie kann aus beliebig vielen Systemelementen (Aggregation: elements) bestehen.

SystemElement**:** Diese Klasse stellt das eigentliche Systemelement dar. Sie ist das zentrale Ausdrucksmittel der Systemstruktur (dargestellt durch ein Sechseck, vgl. 6.3.1). Durch das Attribut „description“ lässt sich das Systemelement umgangssprachlich beschreiben. Die Assoziation isParentOf beschreibt, ob das Systemelement ein hierarchisches Systemelement ist oder nicht (vgl. 6.9).

SystemElementType **(**enumeration**):** Diese Klasse modelliert, ob es sich bei dem Systemelement um ein Software-Element, ein Hardware-Element oder ein logisches Element handelt. Die Klasse beschreibt einen Datentypen.

Discipline: Diese Klasse modelliert die Zuordnung verschiedener Disziplinen zu den einzelnen Systemelementen, wobei jedes Systemelement mind. einer Disziplin zugeordnet werden muss (Aggregation: „disciplines“).

Durch die Zuweisung des Typs (Klasse: SystemElementType) wird die unter 6.1 festgelegte Anforderung (7), „Sowohl Hardware, als auch Softwareelemente müssen gemeinsam modelliert werden können“ erfüllt.

6.4 Funktionserfüllung

Jedes Systemelement besitzt unterschiedliche Fähigkeiten, wie z.B. die Erzeugung elektrischen Stroms, Erzeugung mechanischer Kraft oder die Regelung bewegter Massen. Um diese Fähigkeiten einheitlich modellieren zu können bedarf es einer entsprechenden Systematik. Ziel dieser Systematik ist es, die Fähigkeiten der Systemelemente so zu beschreiben, dass sie miteinander verglichen und gezielt nach ihnen gesucht werden kann. Um dies zu erreichen wird die Funktionsmodellierung aus Kapitel 5 verwendet.

Um die vorgesehene Suche nach Systemelementen überhaupt zu ermöglichen, ist es erforderlich den Systemelementen diejenigen Funktionen zuzuweisen, die sie realisieren können. Diese Zuweisung muss ebenso wie die Festlegung der zu verwendenden Substantive und Verben, im Vorfeld des Systementwurfs erfolgen.

Zu diesem Zweck werden jedem Systemelement diejenigen Funktionen, in Form der in Kapitel 5 beschriebenen Form, zugewiesen, die sie nach Meinung der Entwickler erfüllen können. Die so beschriebenen Systemelemente werden dann in einer *Systemelementbibliothek* abgelegt.

Neben der Angabe, welche Funktionen das Systemelement realisiert, muss zusätzlich noch angegeben werden, welche Hilfsfunktionen es benötigt. Nur so kann nach Auswahl des Systemelements die Funktionshierarchie entsprechend erweitert werden.

Als Beispiel sei ein Hydraulikzylinder genannt, der in der Lage ist eine Bremskraft zu erzeugen. Die umgangssprachliche Beschreibung lautet „Bremskraft erzeugen“ und die Substantiv-Verb-Substantiv Beschreibung ist „--- | erzeugen | Bremskraft“. Vergleiche hierzu das Beispiel aus Kapitel 4, Abbildung 34.

Durch die Zuweisung von Funktionen zu Systemelementen wird die unter 6.1 festgelegte Anforderung (4), „Es muss möglich sein festzulegen, dass ein Systemelement in der Lage ist Funktionen zu realisieren“ erfüllt.

6.4.1 Formalisierung

Das folgende UML-Klassendiagramm (vgl. Abbildung 47) beschreibt die Struktur für die Zuweisung von Funktionen zu Systemelementen. Die Klassen VerbName, VerbGroup, SubjectGroup, SubjectName und Systemelement sind bereits aus Kapitel 5.10 bzw. 6.32 bekannt und werden daher nicht weiter beschrieben. Sie sind lediglich der Vollständigkeit und zum besseren Verständnis hier aufgeführt.

Kurzbeschreibung der Klassen:

VerbName: siehe Kapitel 5.10

VerbGroup: siehe Kapitel 5.10

SubjectName: siehe Kapitel 5.10

SubjectGroup: siehe Kapitel 5.10

Systemelement: siehe Kapitel 6.32

SE_Function: Diese Klasse beschreibt die Funktionserfüllung eines Systemelements. Jedes Systemelement ist in der Lage mehrere Funktionen zu erfüllen (Assoziation: fullfills). Darüber hinaus können über die Assoziation requires die zugehörigen Hilfsfunktionen ermittelt werden (vgl. Abschnitt 6.5). Das zugehörige Verb wird durch die Assoziation selectedVerb spezifiziert und die zugehörigen Input- und Outputsubstantive über die Assoziationen selectedInput und selectedOutput.

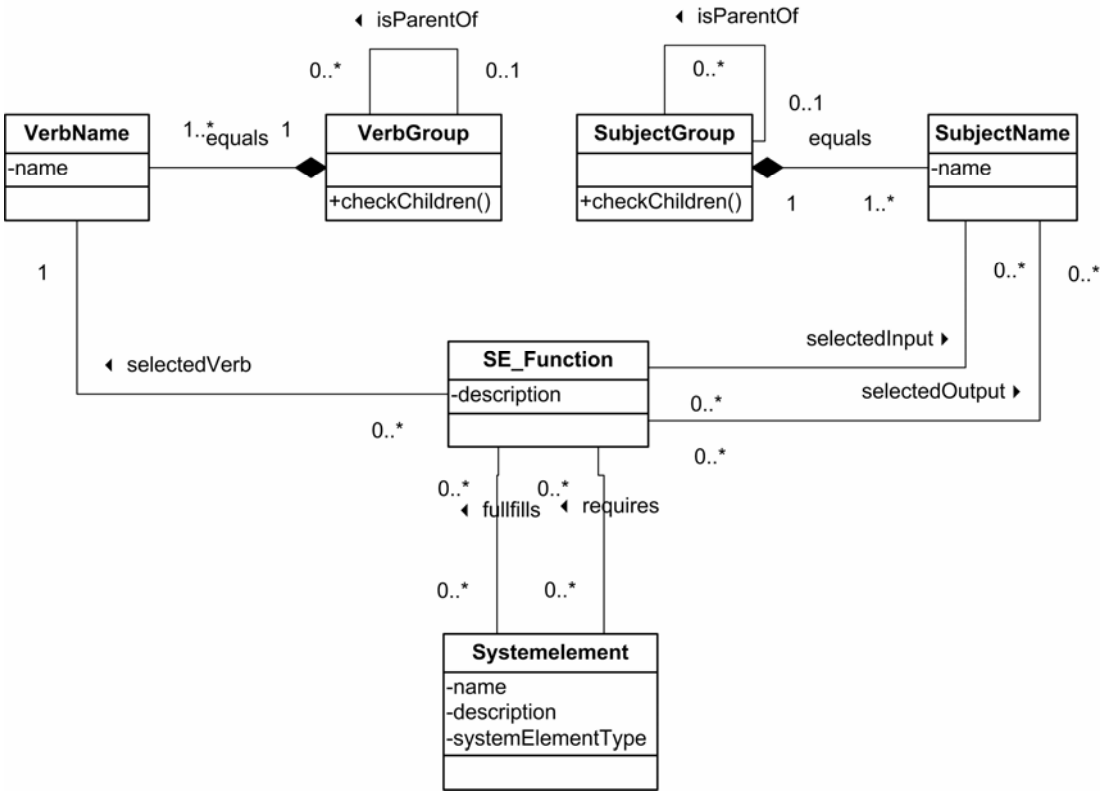


Abbildung 47: Klassendiagramm zur Beschreibung der Realisierungsmöglichkeiten

6.5 Abhängigkeiten

Der Einsatz von Systemelementen innerhalb eines mechatronischen Systems hängt von vielen unterschiedlichen Faktoren ab. Hierzu zählen beispielsweise der vorhandene Bauraum, das verwendete Material, die möglicherweise auftretenden Umwelteinflüsse (Schmutz, Regen, Hitze, Kälte usw.), die zur Verfügung stehende Rechenkapazität oder das zur Verfügung stehende Betriebssystem.

Viele Systemelemente, die in einem mechatronischen System eingesetzt werden, können nur dann verwendet werden, wenn gleichzeitig zusätzliche Systemelemente mit anderen Funktionalitäten vorhanden sind. So funktioniert beispielsweise ein Hydraulikzylinder nicht ohne eine Pumpe die den entsprechenden Volumenstrom bereitstellt, oder der Modulregler lässt sich nicht ausführen, wenn kein entsprechendes Steuergerät inkl. des entsprechenden Betriebssystems verfügbar ist.

Solche Abhängigkeiten werden im Rahmen dieser Arbeit als Beziehung zwischen einem Systemelement und einer oder mehreren Funktionen beschrieben. Das bedeutet, dass ein Systemelement nur dann verwendet werden kann, wenn für die Funktion(en) von dem/denen

es abhängt, ebenfalls entsprechende Systemelemente vorhanden sind. Funktionen von denen ein Systemelement abhängt werden *Hilfsfunktionen* genannt (vgl. Kapitel 5.7).

Durch diese Form von Abhängigkeiten kann es zu einer ganzen Reihe von Folge-Anhängigkeiten kommen. Systemelemente, die zur Realisierung einer Hilfsfunktion ausgewählt wurden, sind selbst wieder von anderen Hilfsfunktionen abhängig. Auf diese Weise entsteht ein ständiger Wechsel zwischen der Systemstruktur und der Funktionshierarchie. Diese Problematik wird in Kapitel 7 noch ausführlicher erläutert.

6.5.1 Zuweisung der Abhängigkeiten

Um festzulegen von welchen Funktionen ein Systemelement abhängig ist, gibt es prinzipiell zwei Möglichkeiten, eine manuelle und eine automatische. Für eine automatische Ermittlung von Hilfsfunktionen muss zum einen jedoch eine Reihe von Informationen über das Systemelement bekannt sein und zum anderen muss ein Algorithmus oder eine Methode entwickelt werden, die aus diesen Informationen auf die Fähigkeiten des Systemelements schließt. Da die Entwicklung eines solchen Algorithmus nicht Gegenstand dieser Arbeit ist, wird die manuelle Zuweisung der Funktionen gewählt.

Immer wenn ein Systemelement spezifiziert und in der Systemelementbibliothek abgelegt wird, dann wird durch den Entwickler festgelegt, welche Hilfsfunktionen es zusätzlich benötigt. Die Angabe der Hilfsfunktionen erfolgt dabei gemäß der in Kapitel 5 beschriebenen Art zur Beschreibung von Funktionen.

Durch die die Angabe von Abhängigkeiten wird die unter 6.1 festgelegte Anforderung (5), „Mögliche Abhängigkeiten von Hilfsfunktionen müssen berücksichtigt werden“ erfüllt.

6.5.2 Formalisierung

Die formale Beschreibung der Abhängigkeiten wird durch das UML-Klassendiagramm in 6.4.1 beschrieben. Die Zuweisung erfolgt durch die Assoziation requires zwischen den Klassen Systemelement und SE_Function.

6.6 Attribute

Welche Funktionen ein Systemelement erfüllt und von welchen es abhängig ist wird in Abschnitt 6.4 bzw. 6.5 beschrieben. Diese Angaben alleine reichen jedoch nicht aus, um ein Systemelement zu charakterisieren. Viele Systemelemente besitzen noch weitere, individuelle Merkmale (z.B. Abmaßen, Messbereichen, Gewicht oder Preis). Diese individuellen Merkmale beschreiben ein Systemelement wesentlich detaillierter und geben einen besseren Überblick darüber, welche zusätzlichen Eigenschaften es besitzt.

6.6.1 Ermittlung von Attributen

Die Ermittlung bzw. Festlegung von Attributen ist nicht trivial, da der Abstraktionsgrad der Systemelemente sehr unterschiedlich sein kann. Dies reicht von allgemeinen Systemelementen wie „Zylindern“ oder „Reglern“, die eher Klassen oder Kategorien darstellen, bis hin zu konkreten Kaufteilen bzw. Lösungselementen mit detaillierten Angaben (z.B. aus Datenblättern).

Im Rahmen des Systementwurfs wird davon ausgegangen, dass eine abstrakte Beschreibung der Systemelemente ausreicht. Zu einem so frühen Zeitpunkt im Entwicklungsprozess wird es meistens noch nicht möglich sein genau zu spezifizieren wie schwer ein Systemelement sein soll, oder in welchem exakten Messbereich es funktionieren soll. Aus diesem Grund werden die Attribute auf einem sehr hohen Abstraktionsniveau spezifiziert. Beispiele hierfür sind

„Anzahl der Freiheitsgrade“, die „Bewegungsrichtung“, die „Betriebsart“ oder die „Übertragungsform“.

Attribute sollen insbesondere dazu dienen Systemelemente genauer zu beschreiben. Zu diesem Zweck besitzt jedes Attribut einen *Namen*, einen *Wert* und falls erforderlich eine *Einheit*. Berechnungen auf Basis der Attribute werden im Rahmen dieser Arbeit nicht durchgeführt.

6.6.2 Attributsbeschreibung

Name: Jedes Attribut hat einen vordefinierten Namen, wie z.B. „Freiheitsgrad“, „Bewegungsrichtung“ oder „Temperaturbereich“.

Wertetyp: Bei den Werten eines Attributs gibt es sowohl numerische als auch nicht-numerische Werte. Die nicht-numerischen Werte werden in Form von vordefinierten Listen zur Verfügung gestellt, aus der dann genau ein Wert ausgewählt wird. Hierbei spricht man von Einzelwerten. Die Auswahl mehrerer Werte aus der vorgegebenen Liste ist nicht möglich.

Bei den numerischen Werten gibt es zwei Wertetypen, einen Einzelwert oder ein Intervall. In beiden Fällen existiert jedoch keine vorgegebene Liste. Es obliegt dem Entwickler im jeweiligen Fall die Werte festzulegen. Es wird lediglich vorgegeben, ob genau ein Wert oder ein Intervall (obere und untere Grenze) angegeben werden muss.

Schließlich ist es möglich, dass ein Attribut eine Einheit besitzt. Dies wird ebenfalls im Vorfeld festgelegt. Bei der Spezifizierung des Attributs ist dann darauf zu achten, dass die Einheit entsprechend passt. Beispielsweise wird das Attribut „Temperaturbereich“ immer in Grad Celsius gemessen und nicht in Fahrenheit. Sollte der Wert nur in Fahrenheit vorliegen, muss er vorher vom Entwickler in Grad Celsius umgerechnet werden.

Durch die Angabe von Attributen wird die unter 6.1 festgelegte Anforderung (3), „Die Angabe von Merkmalen bzw. Eigenschaften für jedes Systemelement muss möglich sein“, erfüllt.

Beispiel

Das folgende Beispiel (vgl. Abbildung 48) verdeutlicht, wie dem Systemelement „Zylinder“ das Attribut „Bewegungsrichtung“ zugewiesen wird. Aus einer Liste vorgegebener Werte (Ausprägungen für Bewegungsrichtung) wird die Ausprägung „Translatorisch“ gewählt. Gleichzeitig hat das Attribut den Wertetyp „Einzelwert“ und besitzt keine Einheit.

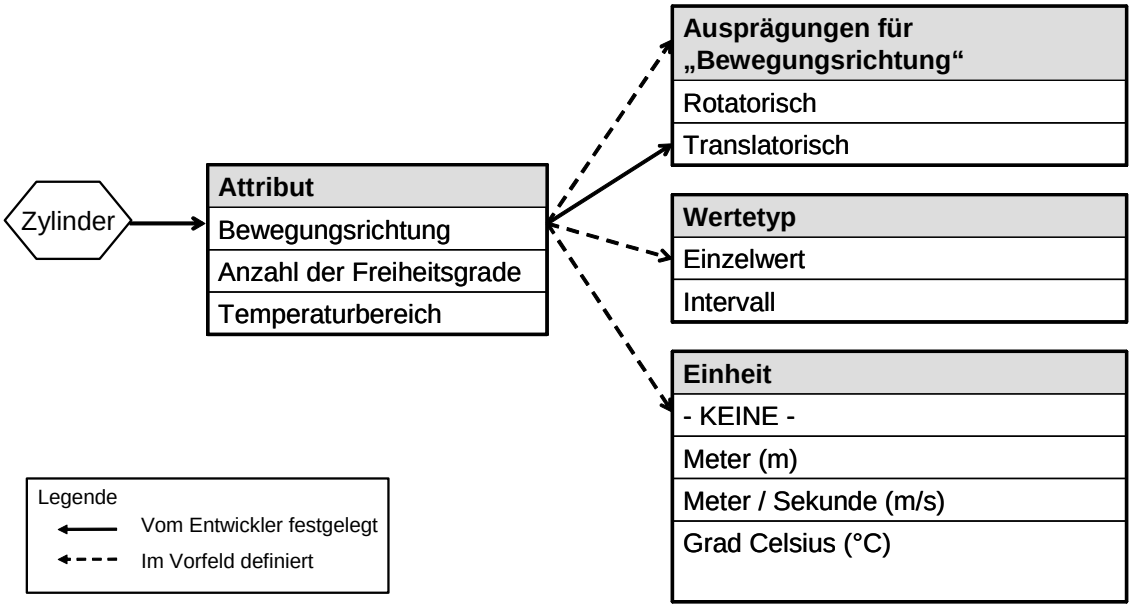


Abbildung 48: Zuweisung von Attributen und Ausprägungen (Schematische Darstellung)

6.6.3 Formalisierung

Abbildung 49 zeigt das UML-Klassendiagramm zur Beschreibung der Attribute, ihrer Werte und Einheiten. Das Diagramm ist aufgeteilt in zwei Bereiche. Auf der linken Seite sind die Klassen zu finden, die die Menge aller Attribute, deren möglichen Ausprägungen, Typen und Einheiten beschreiben. Auf der rechten Seite sind die Klassen zu finden, mit denen die Attribute einem konkreten Systemelement zugewiesen werden.

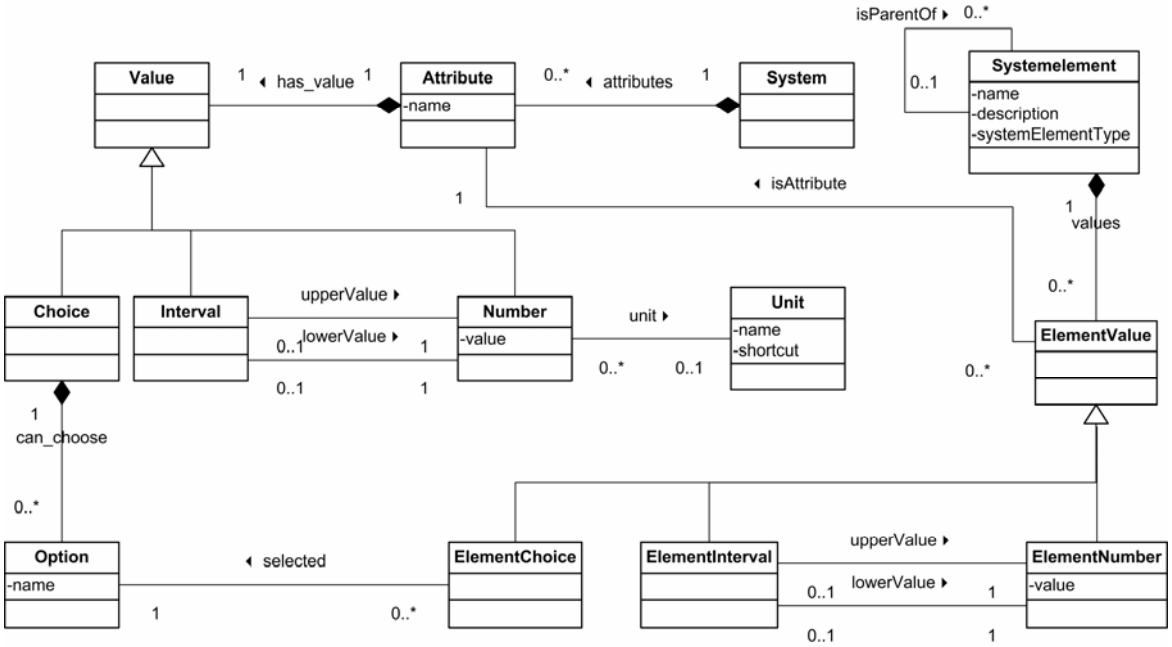


Abbildung 49: UML-Klassendiagramm zur Beschreibung von Attributen

Kurzbeschreibung der Klassen:

System: Siehe 6.3.2.

Attribute: Diese Klasse beschreibt die Attribute, die zur Beschreibung eines Systemelements zur Verfügung stehen. Durch die Variable „name“ erhält das Attribut sine Bezeichnung (z.B.: „Freiheitsgrad“ oder „Temperaturbereich“).

Value (abstract): Diese Klasse stellt den Wertetyp des Attributs dar.

Number: Diese Klasse erbt von der Klasse Value und stellt eine konkrete, einzelne Zahl dar. Die Zahl wird durch die Variable „value“ angegeben. Ein Objekt dieser Klasse kann selbst Teil eines Intervalls sein (vgl. Klasse Interval).

Interval: Diese Klasse stellt ein Intervall von numerischen Werten dar. Die obere (Assoziation: upperValue) bzw. untere Grenze (Assoziation: lowerValue) wird durch die Klasse Number beschrieben.

Choice: Diese Klasse beschreibt eine Liste möglicher Werte. Die Menge der auswählbaren Werte wird durch die 1..n-Composition can choose zur Klasse Option ausgedrückt, d.h. es können beliebig viele Instanzen der Klasse Option erzeugt werden aus denen dann eine ausgewählt wird (vgl. Klasse ElementChoice).

Option: Diese Klasse beschreibt einen Wert einer Liste.

Unit: Diese Klasse beschreibt die Einheit eines numerischen Wertes. Die Variable „name“ gibt der Einheit einen Namen (z.B.: „Grad Celsius“) und mit der Variablen „shortcut“ lässt sich das entsprechende Zeichen (z.B.: „°C“) definieren.

ElementValue (abstract): Diese Klasse beschreibt den Wert eines Attributs für ein konkretes Systemelement. Durch die Assoziation isAttribute lässt sich der Name des Attributes ermitteln.

ElementNumber: Diese Klasse stellt einen numerischen Wert eines Attributs für ein konkretes Systemelement dar.

ElementInterval: Diese Klasse stellt ein Intervall von numerischen Werten für ein konkretes Systemelement dar. Die einzelnen Werte werden mit Hilfe der Assoziationen lowerValue und upperValue erreicht.

ElementChoice: Diese Klasse beschreibt einen aus einer Liste ausgewählten Wert für ein konkretes Systemelement. Die Auswahlliste lässt sich über die Assoziation selected erreichen.

6.7 Ports

In der Arbeit von Kallmeyer werden Systemelemente mit Hilfe der Flussarten Energie-, Stoff- oder Informationsfluss miteinander verknüpft (graphisch dargestellt durch verschiedene Pfeilarten). An diese Flussarten können zusätzliche Informationen bzgl. der transferierten Werte annotiert werden, wie z.B. „Drehzahl“, „Leistung“ oder „Kraft“.

Welche Verbindung im konkreten Fall erstellt wird, entscheidet der Entwickler auf Basis seiner Erfahrung bzw. seines Wissens. Es gibt keine Regeln die angeben, welche Systemelemente durch welche Flussarten miteinander verknüpft werden können.

Darüber hinaus lässt sich nicht feststellen, ob die Systemelemente korrekt verbunden wurden. Korrekt heißt in diesem Fall, dass jeder Input und jeder Output der Systemelemente (jede Schnittstelle) mit einem anderen Systemelement verbunden ist. Somit kann es vorkommen,

dass Schnittstellen unverbunden bleiben und die Systemstruktur unvollständig spezifiziert wird.

Aus diesen genannten Gründen wurden im Rahmen dieser Arbeit so genannte *Ports*[23] für die Beschreibung von Schnittstellen eingeführt. Eine Verbindung zwischen zwei Systemelementen erfolgt über diese Ports.

Ports ermöglichen es zum einen festzulegen, was zwischen Systemelementen übertragen wird und zum anderen helfen sie dabei vollständige Systemstrukturen zu modellieren. Existiert innerhalb der Systemstruktur ein Systemelement mit einem Port der noch nicht verbunden ist, kann der Entwickler hierauf entsprechend aufmerksam gemacht werden.

Um eine Überprüfung zu ermöglichen, wurden bestimmte Eigenschaften definiert die einen Port charakterisieren. Folgende Eigenschaften besitzt ein Port:

- Porttyp
- Flussart
- Verfeinerung der Flussart

6.7.1 Porttyp

Ports werden grundsätzlich in sendende (Outputport) und empfangende Ports (Inputports) unterschieden. Ein bidirektionaler Port, also ein Port der sowohl senden als auch empfangen kann, ist nicht zulässig.

Die Angabe des Porttyps ist Pflicht.

6.7.2 Flussart

In Kapitel 3.1.1 wurde gezeigt, dass Funktionen einer Funktionsstruktur mit Hilfe der drei allgemeinen Größen (Flussarten) *Energie, Stoff* und *Information* miteinander verbunden werden. Diese Art der Verbindung lässt sich auf die Systemstrukturmodellierung übertragen, da Systemelemente prinzipiell Funktionen erfüllen und somit auch die gleichen Flussarten übertragen. Im Detail drückt sich dies darin aus, dass jeder Port eines Systemelements genau eine der drei Flussarten überträgt.

Die Angabe der Flussart ist Pflicht.

Durch die Zuweisung der Flussart wird die unter 6.1 festgelegte Anforderung (2), „Systemelemente müssen mit Hilfe unterschiedlicher Flussarten miteinander verknüpft werden können“ erfüllt.

6.7.3 Verfeinerungen der Flussarten

Die ausschließliche Angabe von Flussarten die ein Port übertragen kann, ist in vielen Fällen zu allgemein. Die Tatsache das Energie, Stoff oder Information übertragen wird ist zu vage und ungenau (vgl. [PB03, S. 220]).

Als Beispiel sei hier ein Hydraulikzylinder genannt, der mit einer Pumpe verbunden werden soll. Bei der Verbindung dieser Elemente wäre es sinnvoll, wenn neben der Angabe das ein Stoff übertragen wird, noch konkretere Angaben gemacht werden können, wie z.B. die Angabe das „Öl“ übertragen wird und nicht „Wasser“ oder „Gas“.

[23] Die Verwendung von Ports basiert auf dem Komponentendiagramm der UML bzw. dem Assemblydiagramm der SysML.

Um dieser Anforderung gerecht zu werden, können die Flussarten weiter verfeinert werden. Die Angabe dieser Verfeinerungen ist optional. Werden allerdings Verfeinerungen angegeben, dann ist nur die Angabe einer einzigen Verfeinerung zulässig.

Energie

Verfeinerung: Energieart

Energie kann ich unterschiedlichen Formen auftreten. Die bekannteste Form ist sicherlich die elektrische Energie. Es gibt aber insbesondere im Bereich der mechanischen bzw. mechatronischen Systeme noch weitere *Energiearten* die von Systemelementen erzeugt und an andere Systemelemente übertragen werden. Mit Hilfe der Energieart lässt sich somit angeben, welche Art von Energie übertragen wird.

Beispielhaft seien folgende Energiearten genannt: elektrische Energie, mechanische Energie, Spannung, Strom, Ladung, Kraft, Impuls, kin. Energie, Moment, Drall, elektrische Leistung, mechanische Leistung, magnetische Energie, Radioaktive Strahlung usw.

Stoff

Verfeinerungen: Stoffart

Beim Stoff verhält es ähnlich wie bei der Energie. Es ist von besonderem Interesse, welche Art von Stoff eigentlich übertragen wird. Dies kann bei der eigentlichen Entwicklung von entscheidender Bedeutung sein. Mit Hilfe von Öl lässt sich beispielsweise wesentlich genauer ein bestimmter Druck erzeugen und regulieren, als mit Wasser oder Luft.

Beispiele für Stoffarten sind: Gas, Wasser, Öl, Benzin, Flüssigkeit, Pulver, Staub, Holz etc.

Information

Bei der Übertragung einer Information stehen insbesondere der Zweck der Information und der Inhalt der Information im Vordergrund.

Verfeinerung: Informationsart

Die Informationsart gibt den genauen Zweck an, wofür die Information genutzt werden soll bzw. wodurch sie ermittelt wurde.

Beispiele für Informationsarten sind: Messwert, Stellwert, Sollwert, Daten, Istwert etc.

Verfeinerung: Informationsinhalt

Der Informationsinhalt beschreibt den Ursprung der Information, also worüber die Information vorliegt. Im Falle eines Messwertes gibt der Informationsinhalt an, dass es sich beispielsweise um eine gemessene Geschwindigkeit oder eine Beschleunigung handelt.

Beispiele für Informationsinhalte sind: Druck, Geschwindigkeit, Beschleunigung, Füllstand, Kraft, Gewicht, Winkel etc.

6.7.4 Beispiel

Das folgende Beispiel (vgl. Abbildung 50) zeigt das Systemelement „Pumpe“ mit vier Ports. Der linke, obere Port ist ein Port vom Typ „Input-Port“ und überträgt die Flussart „Information“. Bei der Informationsart handelt es sich um einen „Stellwert“ und beim Informationsinhalt dreht es sich um Angaben über einen „Volumenstrom“.

Die Eigenschaften der anderen Ports erklären sich entsprechend.

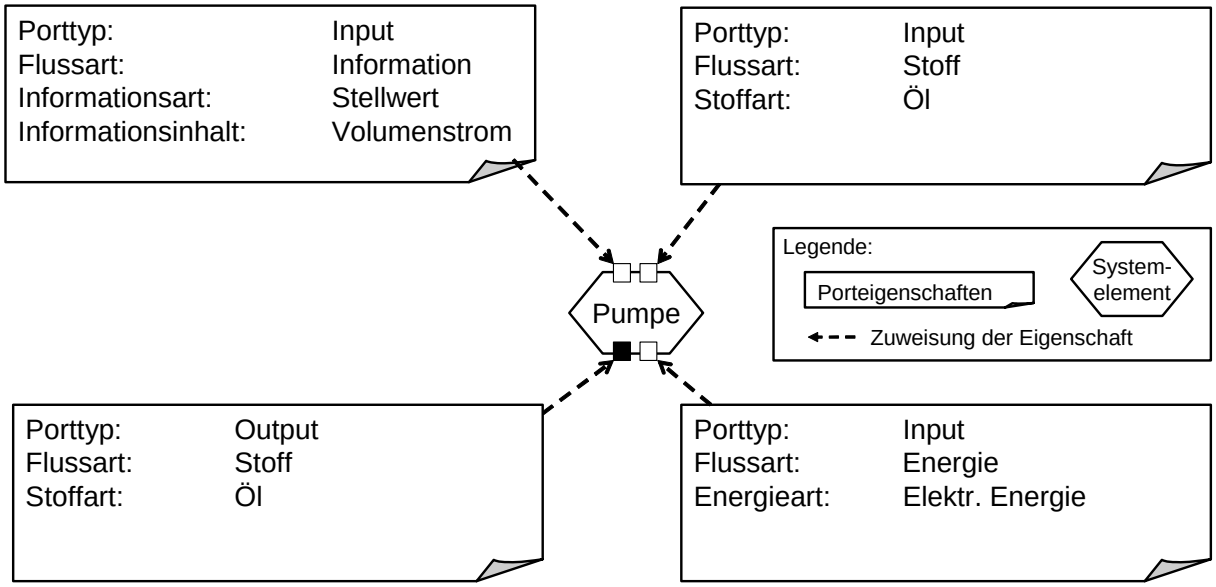


Abbildung 50: Eigenschaften eines Ports am Beispiel des Systemelements „Pumpe“

6.7.5 Notation

Ein Port wird als Quadrat auf dem Rand des Systemelements platziert. Ein Outputport wird als schwarzes Quadrat angezeigt und ein Inputport als weißes Quadrat. Die Eigenschaften werden dabei nicht angezeigt.

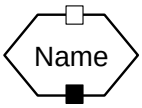


Abbildung 51: Graphische Darstellung eines Systemelements mit Port

Durch die Notation der Systemelemente inkl. der Ports wird die unter 6.1 festgelegte Anforderung (1), „Die Systemstruktur muss graphisch modelliert werden können und sollte für Entwickler der verschiedenen Disziplinen leicht verständlich sein“, erfüllt.

6.7.6 Formalisierung

Die formale Beschreibung der statischen Struktur zeigt das UML-Klassendiagramm in Abbildung 52.

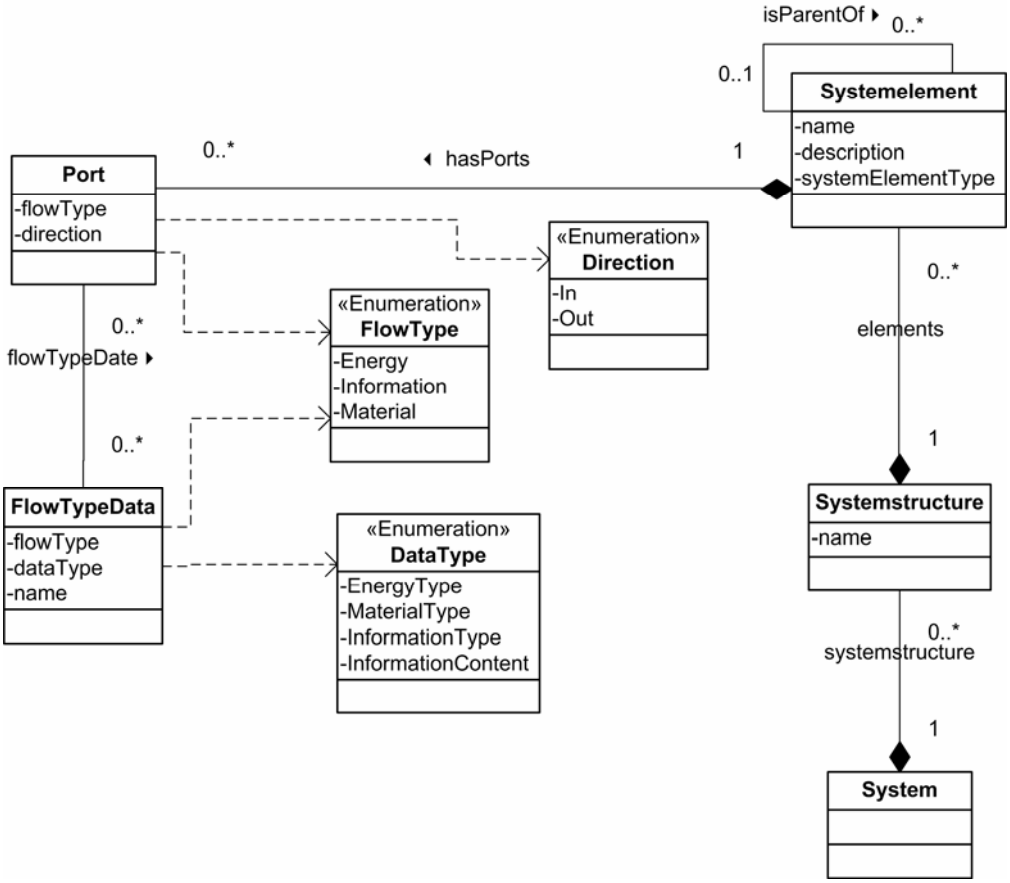


Abbildung 52: UML-Klassendiagramm zur Beschreibung von Ports

Kurzbeschreibung der Klassen:

System: Siehe 6.3.2

Systemstructure: Siehe 6.3.2

Systemelement: Siehe 6.3.2

Port: Diese Klasse beschreibt den eigentlichen Port. Ein Systemelement kann beliebig viele Ports besitzen (vgl. Assoziation hasPorts). Durch die Datentypen FlowType und Direction wird die Flussart und der Porttyp festgelegt. Durch die Assoziation flowTypeDate lassen sich die Verfeinerungen der Flussart spezifizieren.

FlowType (Enumeration): Mit Hilfe dieser Klasse wird beschrieben, welche Flussart mit Hilfe des Ports übertragen wird.

Direction (Enumeration): Mit Hilfe dieser Klasse lässt sich bestimmen, ob es sich bei einem Port um einen Input- oder einen Outputport handelt.

FlowTypeData: Die Klasse modelliert die Verfeinerung der Flussarten. Durch die Datentypen FlowType und DataType lässt sich festlegen zu welcher Flussart die Verfeinerung gehört.

DataType (Enumeration): Diese Klasse beschreibt die Verfeinerung der Flussart. Dies sind entweder die Energieart, die Stoffart, die Informationsart oder der Informationsinhalt.

6.8 Verbindungen

Im letzten Abschnitt wurde gezeigt, wie die Schnittstellen eines Systemelements beschrieben werden. In diesem Abschnitt wird nur gezeigt, wie die Systemelemente mit Hilfe dieser Schnittstellen (Ports) miteinander verbunden werden.

Wie bereits im letzten Abschnitt erläutert, dürfen die Systemelemente nicht beliebig miteinander verknüpft werden. Es muss darauf geachtet werden, dass nur kompatible Ports miteinander verknüpft werden. Nur auf diese Weise ist es möglich korrekte Systemstrukturen zu modellieren.

Die Kompatibilität zweier Ports wird wie folgt festgelegt:

(1) Die Porttypen müssen unterschiedlich sein (Ausnahme siehe 6.9: „Hierarchische Systemelemente“).

(2) Die Flussarten müssen gleich sein.

(3) Wurden bei beiden Ports Verfeinerungen angegeben, dann müssen diese Verfeinerungen gleich sein.

Um die Übersichtlichkeit bei der Erstellung der Systemstruktur nicht zu verlieren ist es zulässig, dass ein einzelner Port mit mehreren anderen Ports verbunden wird, solange die genannten Kompatibilitätskriterien eingehalten werden.

6.8.1 Übertragungsart

Mit Hilfe der bisher beschriebenen Ports und Verbindungen lässt sich spezifizieren, „Was“ zwischen zwei Systemelementen übertragen wird. Diese Angabe reicht allerdings nicht immer aus. So kann insbesondere die Übertragungsart erheblichen Einfluss auf die Systemstruktur haben. Durch Angabe der Übertragungsart wird die Frage nach dem „Worin“ bzw. „Womit“ beantwortet.

Beim der Entwicklung mechatronischer Systeme kann es durchaus einen Unterschied machen, ob beispielsweise eine „Information“ per Funk oder durch ein Kabel übertragen wird. Im letzteren Fall könnte dies Auswirkung auf die Bauform des Systems haben, da Kabelkanäle berücksichtigt werden müssten.

Aufgrund dieses Sachverhalts ist es nicht nur wichtig die übertragene Flussart anzugeben, sondern auch die Art der Übertragung. Dadurch wird es möglich bereits während des Systementwurfs festzuhalten wie die Verbindungen physikalisch aufgebaut werden sollen.

Um dieser Anforderung gerecht zu werden, lässt sich für jede Verbindung angeben, auf welche Art die Übertragung erfolgen soll. Dies erfolgt im Rahmen dieser Arbeit in Form eines frei wählbaren Textes. Eine Verknüpfung mit der Flussart wird nicht gemacht.

6.8.2 Beispiel

Das folgende Beispiel (vgl. Abbildung 53) zeigt einige Systemelemente und deren Verbindungen.

Die Pumpe überträgt durch einen Schlauch (Übertragungsart) Öl (Stoffart) an das Ventil. Gleichzeitig wird vom Ventilregler ein Stellwert (Informationsart) über einen Volumenstrom (Informationsinhalt) durch ein Kabel (Übertragungsart) an das Ventil gesendet. Das Ventil öffnet bzw. schließt sich und überträgt dadurch durch einen Schlauch (Übertragungsart) das Öl (Stoffart) an den Zylinder. Schließlich fährt der Kolben (Übertragungsart) des Zylinders aus und überträgt dadurch eine Kraft (Energieart) an den Träger.

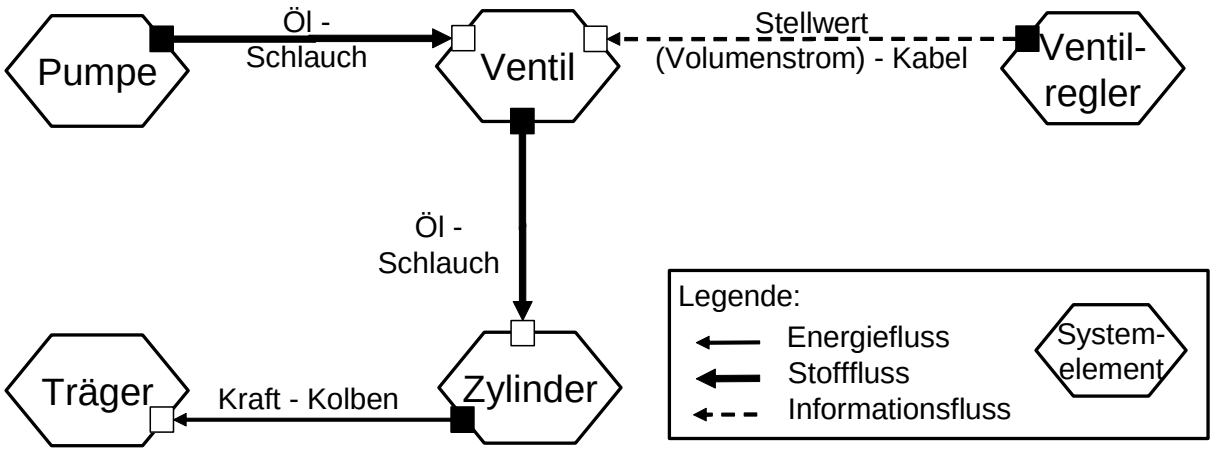


Abbildung 53: Beispiel einer Systemstruktur (Ausschnitt)

6.8.3 Notation

Verbindungen werden durch einen gerichteten Pfeil zwischen zwei Ports dargestellt. Die Art des Pfeils gibt an, ob es sich um einen Energiefluss (durchgezogene Linie, Abbildung 54), um einen Stofffluss (dicke, durchgezogene Linie, Abbildung 55) oder um einen Informationsfluss (gestrichelte Linie, Abbildung 56) handelt.

An den Verbindungen werden die unter 6.7.3 vorgestellten Verfeinerungen angegeben. Die drei Verfeinerungen Energieart, Stoffart und Informationsart stehen zuerst. Informationsinhalten werden in eine Klammer geschrieben. Die Übertragungsart steht hinter den Verfeinerungen durch einen Bindestrich getrennt.

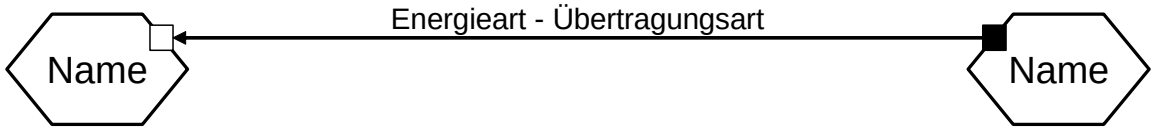


Abbildung 54: Graphische Darstellung eines Energieflusses

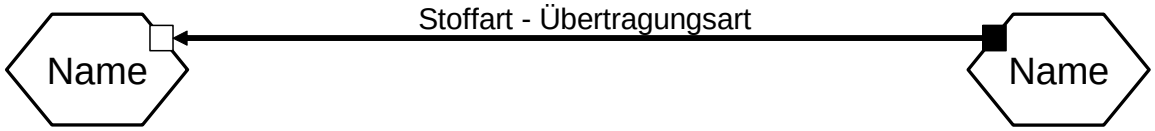


Abbildung 55: Graphische Darstellung eines Stofffllusses

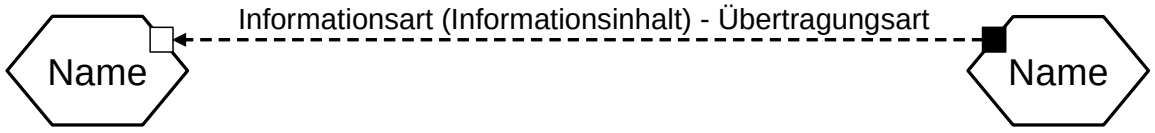


Abbildung 56: Graphische Darstellung eines Informationsflusses

6.8.4 Formalisierung

Folgendes UML-Klassendiagramm beschreibt die Struktur einer Verbindung.

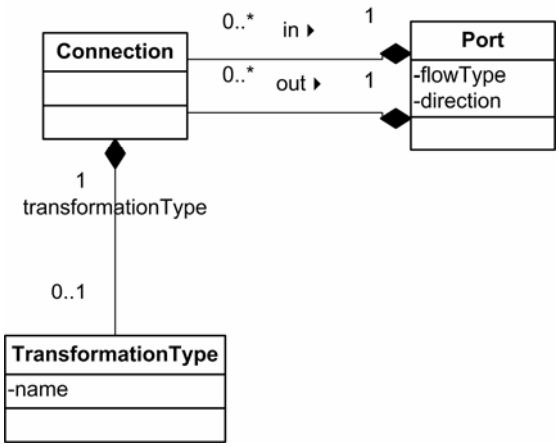


Abbildung 57: UML-Klassendiagramm zur Modellierung von Verbindungen

Kurzbeschreibung der Klassen:

Port: siehe 0

Connection: Diese Klasse modelliert die Verbindung zwischen zwei Ports. Die Assoziationen in und out beschreiben die Verbindung zu den jeweiligen Ports und geben darüber hinaus die Richtung an.

TransformationType: Diese Klasse modelliert die Übertragungsart. Für jede Connection kann angegeben werden (Assoziation transformationType), durch was die Übertragung erfolgen soll. Mit Hilfe des Attributs name wird die Übertragungsart beschrieben (z.B.: „Schlauch“, „Kabel“, „Kolben“, „Funkt“ usw.).

6.9 Hierarchische Systemelemente

In den vorangegangenen Abschnitten wurde die Charakteristik eines einzelnen Systemelements ausführlich vorgestellt. Dabei wurde ein Systemelement als ein in sich abgeschlossenes, *atomares* Element betrachtet.

Die Modellierung mit ausschließlich atomaren Systemelementen reicht jedoch nicht aus. Die dadurch entstehenden Systemstrukturen werden sehr groß und unübersichtlich, wodurch sich keine komplexen Systemstrukturen in überschaubarer und verständlicher Form spezifizieren lassen.

Aus diesem Grund ist es notwendig Systemelemente spezifizieren zu können, die selber wieder aus Systemelementen (*interne Systemelemente*)[24] zusammengesetzt werden, um dadurch die Komplexität zu reduzieren. Darüber hinaus ermöglichen solche *hierarchischen Systemelemente* neue Strukturierungs- und Gruppierungsmöglichkeiten die mit elementaren Systemelementen nicht möglich wären. Beispielsweise lassen sich Systemelemente, die logisch oder physikalisch zusammen gehören, innerhalb eines hierarchischen Systemelements gruppiert.

[24] Systemelemente die Bestandteil eines anderen Systemelements sind, werden als „interne Systemelemente“ bezeichnet.

6.9.1 Hierarchisierung

Hierarchische Systemelemente sind Systemelemente die aus einem oder mehreren anderen Systemelementen bestehen, wobei sie selbst auch wieder Bestandteil anderer hierarchischer Systemelemente sein können. Hierarchische Systemelemente können, ebenso wie atomare Systemelemente, entweder Software-Systemelemente (SW), Hardware-Systemelemente (HW) oder logische Systemelemente (L) sein.

Auf Basis dieser drei Typen (vgl. Klasse SystemElementType aus Abbildung 46) lassen sich neun verschiedene Arten von hierarchischen Beziehungen konstruieren. Hierbei sind allerdings nicht alle Kombination zulässig, d.h. nur bestimmte Typen können miteinander kombiniert werden.

Im folgenden werden die einzelnen Kombinationsmöglichkeiten beschrieben, also welche Systemelemente (von welchem Typ), in welchen hierarchischen Systemelementen (von welchem Typ) platziert werden dürfen.

Der Wert vor dem Trennstrich gibt den Typ des internen Systemelements an, der Wert hinter dem Trennstrich den Typ des hierarchischen Systemelements:

(1) *HW/HW (ZULÄSSIG):* Die Platzierung eines Hardware-Systemelements in einem anderen Hardware-Systemelement ist zulässig. Diese Beziehung spiegelt typischerweise eine physikalische Beziehung (Baustruktur) wieder.

(2) *SW/HW (ZULÄSSIG):* Die Platzierung eines Software-Systemelements in einem Hardware-Systemelement ist zulässig. Das Software-Systemelement wird auf dem Hardware-Systemelement ausgeführt (Beispiel: Regler und Steuergerät).

(3) *SW/SW (ZULÄSSIG):* Die Platzierung eines Software-Systemelements in einem anderen Software-Systemelement ist zulässig (Beispiel: Regler benötigt ein RTOS[25]).

(4) *HW/SW (NICHT ZULÄSSIG):* Die Platzierung eines Hardware-Systemelements in einem Software-Systemelement ist nicht zulässig.

(5) *L/L (ZULÄSSIG):* Die Platzierung eines logischen Systemelements in einem anderen logischen Systemelement ist zulässig (Beispiel: MFM[26] in einem anderen MFM).

(6) *SW/L (ZULÄSSIG):* Die Platzierung eines Software-Systemelements in einem logischen Systemelement ist zulässig (Beispiel: Regler in einem MFM).

(7) *HW/L (ZULÄSSIG):* Die Platzierung eines Hardware-Systemelements in einem logischen Systemelement ist zulässig (Beispiel: Zylinder in einem MFM).

(8) *L/SW (NICHT ZULÄSSIG):* Die Platzierung eines logischen Systemelements in einem Software-Systemelement ist nicht zulässig.

(9) *L/HW (NICHT ZULÄSSIG):* Die Platzierung eines logischen Systemelements in einem Hardware-Systemelement ist nicht zulässig.

6.9.2 Ports und Verbindungen

Ports hierarchischer Systemelemente können im Gegensatz zu Ports atomarer Systemelemente zwei unterschiedliche Aufgaben erfüllen. Zum einen können sie die von dem Systemelement abgegebenen bzw. aufgenommenen Flüsse übertragen. Dabei werden die Flüsse direkt von dem Systemelement verarbeitet bzw. erzeugt. Zum anderen können sie

[25] Real-Time Operating System (Realzeit Betriebssystem)
[26] Mechatronisches Funktionsmodul - Logisches Systemelement bestehend aus Software- und Hardwareelementen

abgegebene bzw. aufgenommene Flüsse auch nur an interne Systemelement weiterleiten, ohne das das Systemelement, zu dem sie gehören, diese Flüsse verarbeitet oder erzeugt hat. Diese Ports werden als leitende Ports bezeichnet und stellen lediglich die Verbindung zwischen einem internen Systemelement und der Außenwelt her.

Jedes interne Systemelement muss über einen leitenden Port mit anderen Systemelementen verbunden werden, wenn diese sich nicht auf gleicher hierarchischer Ebene im gleichen hierarchischen Systemelement befinden.

Anpassung der Kompatibilitätsregeln

Aufgrund der zusätzlichen Eigenschaft gibt es eine Änderung der Kompatibilitätsregeln bei der Erzeugung von Verbindungen. Die Regel (1) aus Abschnitt 6.8 ändert sich wie folgt:

(1) Die Porttypen müssen unterschiedlich sein. Ausnahme: Wird ein leitender Port mit einem Port verbunden der zu einem internen Systemelement gehört, dann müssen die Porttypen gleich sein.

Durch die Modellierung hierarchischer Systemelementen wird die unter 6.1 festgelegte Anforderung (6), „Die Modellierung von hierarchischen Systemelementen muss möglich sein“ erfüllt.

6.9.3 Beispiel

Das folgende Beispiel zeigt das hierarchische Systemelement „Ventil MFM“, bei dem es sich um ein logisches Systemelement (vgl. 6.9.1) handelt. Es enthält die internen Systemelemente „Wegmesser“, „Ventil“ und „Ventilregler“. Der Ventilregler erhält von einem übergeordneten Regler einen Sollwert übermittelt. Gleichzeitig erhält er von dem Wegmesser einen Messwert über den Weg des Ventils und sendet einen Stellwert zum stellen (öffnen und schließen) des Ventils.

Der Wegmesser misst den Weg am Ventil (Öffnung des Ventils) und sendet den gemessenen Wert an den „Ventilregler“.

Das Ventil erhält vom Ventilregler die Vorgabe, wie weit es sich öffnen bzw. schließen soll. Gemäß diesen Vorgaben lässt es die Ölmenge durch.

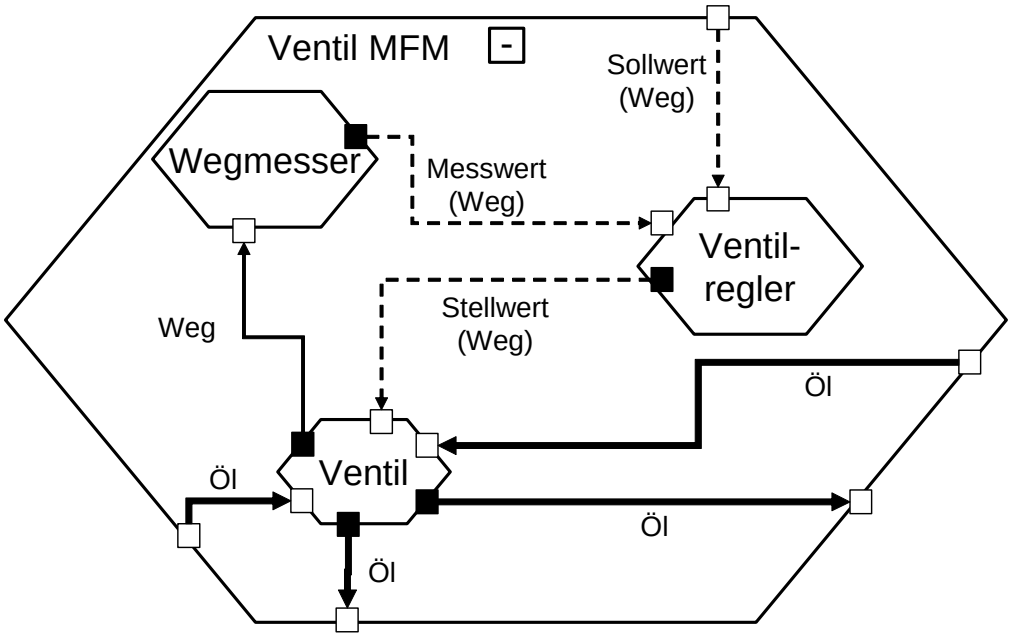


Abbildung 58: Hierarchisches Systemelement (Beispiel)

6.9.4 Notation

Bei hierarchischen Systemelementen wird der Name des hierarchischen Systemelements am oberen Rand dargestellt. Die Ports werden auf dem Rand platziert.

Offener Zustand: Mit einem Minuszeichen wird angezeigt, dass es sich um ein hierarchisches Systemelement handelt, welches sich im offenen („aufgeklappten“) Zustand befindet (vgl. Abbildung 56). In diesem Zustand werden alle internen Systemelemente inkl. ihrer Ports und Verbindungen angezeigt.

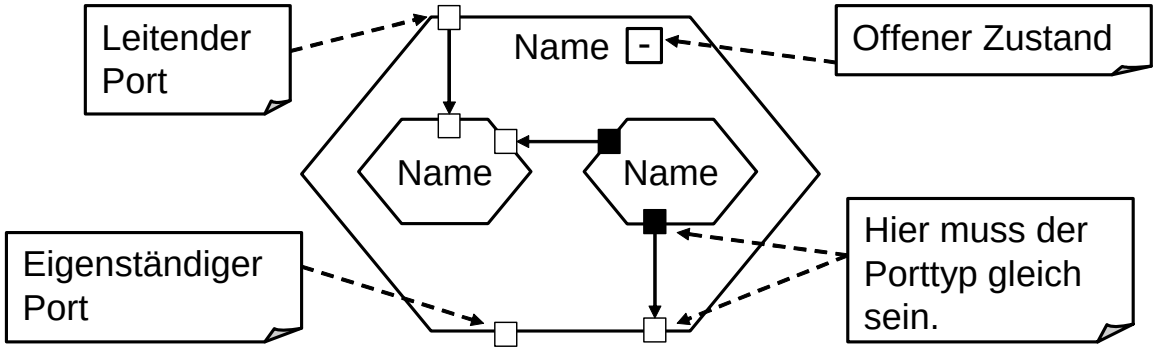


Abbildung 59: Hierarchisches Systemelement (offen)

Geschlossener Zustand: Mit einem Pluszeichen wird angezeigt, dass es sich um ein hierarchisches Systemelement handelt, welches sich im geschlossenen („zugeklappten“) Zustand befindet (vgl. Abbildung 57). Der Name des Elements wird in der Mitte dargestellt und die internen Systemelemente werden nicht angezeigt. Lediglich die Ports des hierarchischen Systemelements werden dargestellt.

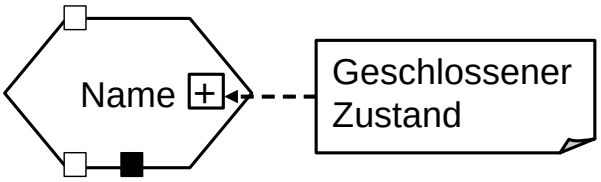


Abbildung 60: Hierarchisches Systemelement (geschlossen)

6.9.5 Formalisierung

Siehe: 6.3.2.

6.10 Zusammenfassung

In diesem Kapitel wird eine Systemstruktur zur Modellierung mechatronischer Systeme vorgestellt. Die Grundlage hierzu liefert die Arbeit von Ferdinand Kallmeyer [Kal98]. Aufgrund von zusätzlichen Anforderungen, die unter anderem im Rahmen des SFB 614 identifiziert wurden, wurde die Systemstruktur konzeptionell erweitert und mit Hilfe der Unified Modeling Language (UML) formalisiert.

Es wurden neue Konstrukte wird z.B. Ports zur Modellierung der Systemelemente hinzugefügt. Des Weiteren wurden Eigenschaften für die Ports definiert und Kompatibilitätsregeln festgelegt, die angeben, welche Ports verbunden werden können und welche nicht. Mit Hilfe der definierten UML-Klassendiagramme wurde die Grundlage geschaffen, um die Suche nach Systemelementen zu unterstützen (vgl. Kapitel 7) und um die Konsistenz zur Funktionshierarchie gewährleisten zu können (vgl. Kapitel 8).

Systemelemente können unterschiedliche Grade der Konkretisierung aufweisen. Dies reicht von sehr konkreten Systemelementen, die direkt bei Herstellern bezogen werden können[27], bis hin zu sehr abstrakten Systemelementen die im Grunde nur Klassen bzw. Kategorien beschreiben. Die Übergänge sind fließend und lassen sich daher nur sehr schwer trennen. Da der Grad der Abstraktion jedoch keinen direkten Einfluss auf die Systemmodellierung hat, wurde auf eine Einteilung der Systemelemente in verschiedene Abstraktionsgrade verzichtet.

[27] Diese sehr konkreten Systemelemente werden auch als Kaufteile oder Lösungselemente bezeichnet.

KAPITEL 7: SYSTEMELEMENTSUCHE

In Kapitel drei wurden einige Schwachstellen der derzeitigen Modellierung mechatronischer Systeme aufgezeigt. Zwei dieser Schwachstellen sind die Funktions- und Systemstrukturmodellierung, die in den Kapiteln fünf und sechs entsprechend bearbeitet wurden. Eine weitere Schwachstelle ist der systematische Übergang von der Funktionshierarchie zur Systemstruktur. Dieser Übergang wird im Folgenden mit Hilfe eines entsprechenden Algorithmus, auf Basis von Graphtransformationsregeln, realisiert.

Das vorliegende Kapitel beschreibt zuerst die Ausgangssituation und die Anforderungen für die automatische Suche nach Systemelementen. Dann wird der entwickelte Algorithmus erläutert und mit Hilfe von Graphtransformationsregeln formalisiert. Im Anschluss daran wird anhand eines ausführlichen Beispiels die Funktionsweise des Algorithmus erläutert. Zum Schluss wird die Laufzeit des Algorithmus betrachtet und eine Zusammenfassung des Kapitels gegeben.

7.1 Ausgangssituation

Die Ausgangssituation für die Suche nach Systemelementen ist die in Kapitel fünf vorgestellte Funktionshierarchie und die in Kapitel sechs vorgestellte Systemstruktur. Für die in der Funktionshierarchie beschriebenen Funktionen müssen Systemelemente gefunden werden, die diese Funktionen realisieren. Hierbei ist es möglich, dass nicht ein einzelnes, atomares Systemelement (vgl. Kapitel 6.3) die Funktion erfüllt, sondern nur mehrere Systemelemente gemeinsam (vgl. Kapitel 6.9, „Hierarchische Systemelemente“).

Die bisherige Arbeitsweise bei der Spezifizierung von Funktionshierarchien sah vor, dass der Entwickler sich bereits während der Erstellung der Funktionen Gedanken darüber machte, welche Systemelemente diese Funktionen später realisieren sollen (vgl. Kapitel 5.1).

An dieser Stelle setzt der im Folgenden beschriebene Ansatz an. Der Entwickler soll die Möglichkeit haben lösungsneutral Funktionen modellieren zu können. Zu diesem Zeitpunkt des Entwurfs soll der Entwickler nicht darüber nachdenken müssen, ob es eine Lösung gibt, d.h. ob es Systemelemente gibt, die zusammen eine funktionierende Lösung darstellen. Die Beantwortung dieser Frage soll ein entsprechender Suchalgorithmus liefern.

Dieser Algorithmus soll in der Lage sein für eine spezifizierte Funktion alle passenden Systemelemente zu ermitteln, die zur Realisierung der Funktion in Frage kommen. Auf diese Weise können Vorfixierungen deutlich verringert werden, wodurch wiederum neue Lösungen möglich werden.

7.2 Anforderungen an die Systemelementsuche

Bei der Systemelementsuche müssen eine Reihe von Besonderheiten berücksichtigt werden, damit die Suche auch die gewünschten Ergebnisse liefert. Unterschiedliche Aspekte müssen betrachtet werden, die in ihrer Gesamtheit dann die Anforderungen an die Suche darstellen. Folgende Anforderungen sind gegeben:

(1) **Bibliothek von Systemelementen:** Damit eine Suche nach Systemelementen überhaupt Erfolg haben kann, muss eine gewisse Menge an Systemelementen vorhanden sein.

(2) **Systemelemente der Bibliothek müssen Funktionen zugewiesen sein:** Jedem Systemelement in der Bibliothek müssen im Vorfeld bereits Funktionen zugewiesen

sein, damit ein entsprechender Algorithmus nach diesen Funktionen suchen und sie vergleichen kann.

(3) **Hilfsfunktionen:** Die von den Systemelementen benötigten Hilfsfunktionen müssen in der Bibliothek hinterlegt werden können.

(4) **Die Äquivalenzrelationen bei Substantiven und Verben müssen berücksichtigt werden:** Die in Kapitel fünf beschriebene Äquivalenzrelation innerhalb der Substantive und Verben soll bei der Suche nach Systemelementen berücksichtigt werden.
Wird nach einer bestimmten Funktion mit bestimmten Substantiven und Verben gesucht, dann sollen nicht nur die Systemelemente ermittelt werden, die genau diese Funktion erfüllen, sondern auch diejenigen, die Funktionen mit äquivalenten Substantiven und Verben erfüllen.

(5) **Die Konkretisierungsrelation bei Substantiven und Verben müssen berücksichtigt werden:** Die in Kapitel fünf beschriebene Konkretisierungsrelation innerhalb von Substantiven und Verben soll bei der Suche nach Systemelementen berücksichtigt werden.
Wird nach einer bestimmten Funktion mit bestimmten Substantiven und Verben gesucht, dann sollen nicht nur die Systemelemente ermittelt werden, die genau die Funktion erfüllen, sondern auch diejenigen, die Funktionen erfüllen, dessen Substantive bzw. Verben konkreter sind als in der Suche angegeben.

In den folgenden Kapiteln werden die hier beschriebenen Anforderungen detailliert erläutert und aufgezeigt, wie sie im Rahmen dieser Arbeit berücksichtigt werden.

7.3 Bibliothek von Systemelementen

Die Suche nach Systemelementen erfordert es, dass Systemelemente vorhanden sind unter denen eine Suche erfolgen kann. Aus diesem Grund ist als Grundlage für die Systemelementsuche eine Bibliothek von Systemelementen vorgesehen.

In dieser Bibliothek werden vom Entwickler diejenigen Systemelemente gespeichert, die als mögliche Lösungen für Funktionen in Frage kommen. Für jedes Systemelement werden zum einen Daten gespeichert die für die Suche relevant sind (vgl. 7.3.1) und zum anderen Daten gespeichert die für die Modellierung der Systemelemente innerhalb der Systemstruktur von Bedeutung sind (vgl. 7.3.2). Zur letzteren Kategorie gehören beispielsweise Angaben über die Anzahl und Typen der Ports oder die hierarchische Struktur der Systemelemente.

Durch die Nutzung einer Bibliothek wird die unter 7.2 festgelegte Anforderung (1), „Damit eine Suche nach Systemelementen überhaupt Erfolg haben kann, muss eine gewisse Menge an Systemelementen vorhanden sein“ erfüllt.

Hinweis

An dieser Stelle ist zu erwähnen, dass es im Rahmen dieser Arbeit nicht die Aufgabe war eine umfangreiche Bibliothek von Systemelementen aufzubauen. Das eine solche Bibliothek existiert, um den beschriebenen Ansatz nutzen zu können, wird vorausgesetzt bzw. angenommen.

Um den Aufbau einer solchen Bibliothek zu unterstützen wäre es möglich, bereits existierende Softwaresysteme bzw. deren Bibliotheken zu verwenden. Als Beispiel seien die Systeme TechOptimizer (vgl. Kapitel 2.2.2.4) und Schemebuilder (vgl. Kapitel 3.12) genannt. Beide Systeme verfügen über eine Bibliothek von Elementen vergleichbar mit denen in dieser

Arbeit betrachteten Systemelemente. Die konzeptionelle und technische Anbindung müsste allerdings noch erfolgen.

7.3.1 Für die Suche relevante Daten der Systemelemente

Für jedes Systemelement, welches sich in der Bibliothek befindet, müssen im Vorfeld folgende Daten festgelegt werden:

Funktionen, die von dem Systemelement erfüllt werden (vgl. Kapitel 6.4): Die Suche nach Systemelementen mit Hilfe eines Suchalgorithmus erfordert es, dass eine Beziehung zwischen Funktion und Systemelement hergestellt werden kann. Diese Beziehung muss im Vorfeld durch den Entwickler spezifiziert werden, indem er angibt, welche Funktionen von dem Systemelement realisiert werden können. Abbildung 61 zeigt beispielhaft die Zuweisung von zwei Funktionen zu einem Systemelement.

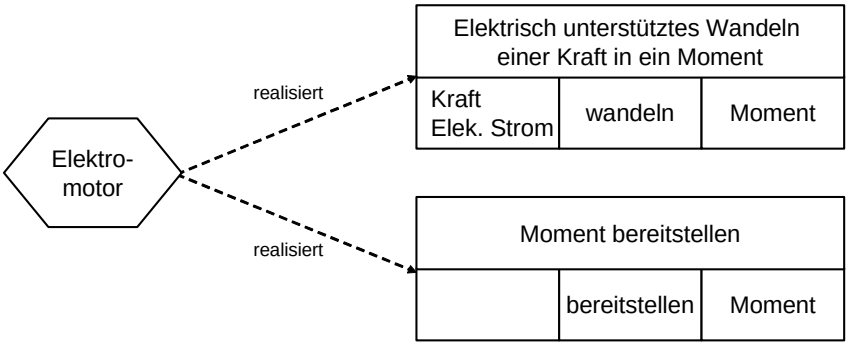


Abbildung 61: Zuweisung von Funktionen zu einem Systemelement

Durch das Speichern der Funktionen zusammen mit den Systemelementen wird die unter 7.2 festgelegte Anforderung (2), „Systemelementen der Bibliothek müssen Funktionen zugewiesen sein“ erfüllt.

7.3.2 Weitere Daten der Systemelemente

Zuzüglich zu den Daten der Systemelemente, die für die Suche notwendig sind, müssen noch weitere Daten gespeichert werden. Hierbei handelt es sich um:

Hilfsfunktionen, die das Systemelement benötigt (vgl. Kapitel 5.7): Es kann vorkommen, das ein Systemelement Hilfsfunktionen benötigt. Diese Hilfsfunktionen müssen für jedes Systemelement explizit angegeben und in der Bibliothek gespeichert werden. Auf diese Weise ist es möglich bei der Wahl des jeweiligen Elements die Funktionshierarchie um die Hilfsfunktionen zu erweitern, um dadurch den Entwickler darauf aufmerksam zu machen, dass hierfür wieder Systemelemente ermittelt werden müssen.

Struktureller Aufbau eines Systemelements: Jedes Systemelement, das sich in der Bibliothek befindet, kann in der Systemstruktur verwendet werden. Um dies zu ermöglichen, müssen jedoch noch weitere Informationen des Elements gespeichert werden. Zu diesen Informationen gehören Angaben über vorhandene Ports oder ob es sich um ein hierarchisches Systemelement handelt und welche internen Systemelemente es besitzt. Diese Topologieinformationen haben keinen Einfluss auf die Suche, sie werden nur der Vollständigkeit halber aufgeführt.

Durch das Speichern der Hilfsfunktionen zusammen mit den Systemelementen wird die unter 7.2 festgelegte Anforderung (3), „Die von den Systemelementen benötigten Hilfsfunktionen müssen in der Bibliothek hinterlegt werden können“ erfüllt.

Attribute eines Systemelements (vgl. Kapitel 6.6): Neben der Zuweisung der Funktionen ist es möglich noch weiteren Merkmale (Attribute) der Systemelemente in der Bibliothek zu speichern. Unter Attributen werden Angabe wie Abmaße, Messbereich, Gewicht oder Preis verstanden die ein Systemelement genauer charakterisieren. Abbildung 62 zeigt beispielhaft die Zuweisung von vier Attributen zu einem Systemelement.

Die Speicherung der Attribute in der Systemelementbibliothek ermöglicht eine genauere Modellierung der Systemstruktur, da detaillierte Daten der Systemelemente berücksichtigt werden können. Einen Einfluss auf die Suche haben diese Werte jedoch nicht.

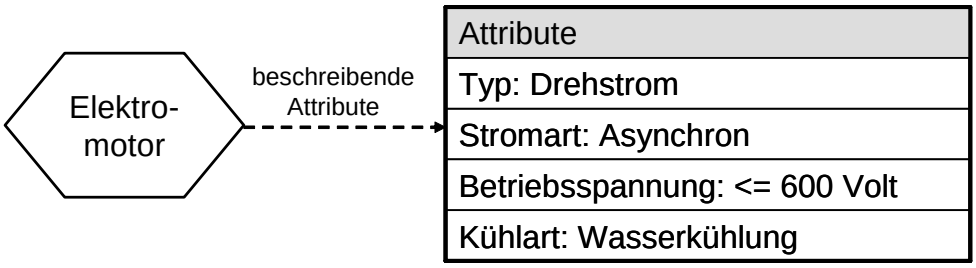


Abbildung 62: Zuweisung von Attributen zu einem Systemelement

7.3.3 Datenmodell

In Abbildung 63 werden in Form eines UML Klassendiagramms diejenigen Klassen gezeigt, die für die Suche nach Systemelementen relevant sind. Einige Klassen wurden bereits in den Kapiteln fünf und sechs erklärt. Neu sind die Klasse SearchResult, RequiredSubjects und SE_Function, wobei die beiden erstgenannten lediglich Hilfsklassen für die Suche darstellen. Nähere Angaben zur Suche befinden sich in Abschnitt 7.4.

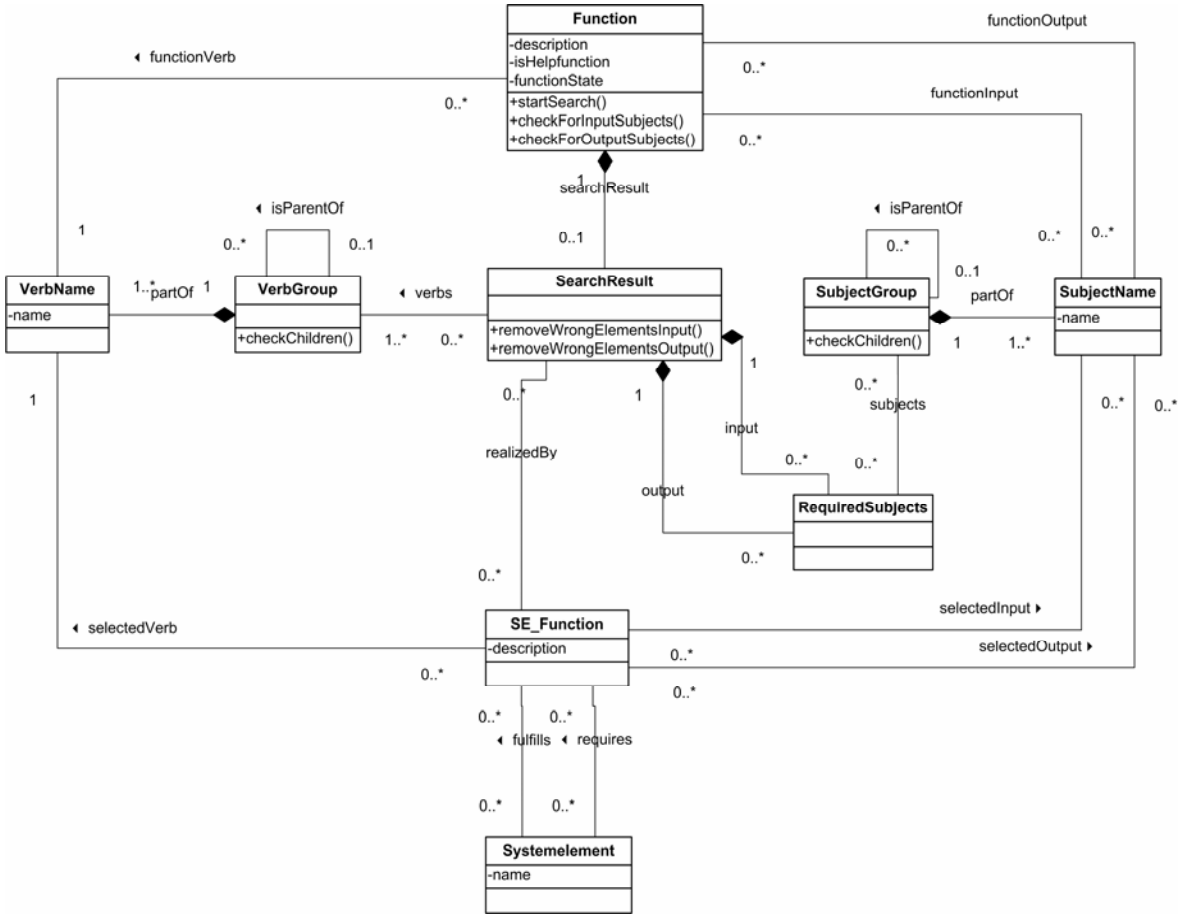


Abbildung 63: Datenmodell zur Formalisierung der Suche

Kurzbeschreibung der Klassen:

Function: siehe 5.10

VerbName: siehe 5.10

VerbGroup: siehe 5.10

SubjectGroup: siehe 5.10

SubjectName: siehe 5.10

Systemelement: siehe 6.3.2

SE_Function**:** Die Klasse SE_Function beschreibt die Funktionen die ein Systemelement prinzipiell realisieren kann. (vgl. Assoziation fulfills). Jedes Systemelement ist in der Lage keine oder beliebig viele Funktionen zu erfüllen. Jede Funktion hat genau ein Verb (Assoziation selectedVerb), mehrere Input-Substantive (Assoziation selectedInput) und mehrere Output-Substantive (Assoziation selectedOutput). Mit Hilfe der Assoziation requires werden die Hilfsfunktionen mit dem Systemelement verknüpft.

SearchResult: Diese Klasse wird benötigt, um das Suchergebnis zu speichern. Zu jeder Suche gehört genau ein Objekt dieser Klasse. Während der Algorithmus abgearbeitet wird, werden andere Objekte an dieses Objekt gelinkt bzw. Links gelöscht. Nach Beendigung des Algorithmus lassen sich über die Assoziation realizedBy bzw. fulfills diejenigen Systemelemente erreichen, die die spezifizierte Funktion erfüllen.

RequiredSubjects**:** Diese Klasse wird bei der Suche benötigt, um die verschiedenen Input- und Output-Substantive zu analysieren und um festzustellen, welche Systemelemente die in der Funktion spezifizierten Input-Substantive bzw. Output-Substantive erfüllen.

7.4 Allgemeine Beschreibung des Suchalgorithmus

Nachdem im vorangegangenen Abschnitt die Voraussetzungen für die Suche nach Systemelementen beschrieben wurden, wird im Folgenden der konkrete Suchalgorithmus erläutert. Dazu wird ein UML-Aktivitätendiagramm verwendet, anhand dessen die einzelnen Schritte erläutert werden (vgl. Abbildung 64).

Schritt 1) Bei Schritt eins handelt es sich um einen Zustand. Dieser stellt die Ausgangssituation dar und besagt, dass für eine Funktion, für die Systemelemente ermittelt werden sollen, alle Input- und Output-Substantive sowie das entsprechende Verb spezifiziert wurden.

Schritt 2) Im zweiten Schritt wird ermittelt, zu welcher äquivalenten Verbgruppe das in der Funktion spezifizierte Verb gehört.

Schritt 3) Im nächsten Schritt werden alle äquivalenten Verbgruppen ermittelt, die Verben enthalten, die das in der Funktion spezifizierte Verb konkretisieren.

Schritt 4) Im vierten Schritt werden alle Systemelemente ermittelt, die das Verb erfüllen, das in der Funktion spezifiziert wurde. Hierbei kann es sich um genau das gleiche Verb handeln, um ein Verb aus der gleichen äquivalenten Verbgruppe oder sogar um ein Verb aus einer Verbgruppe die das gesuchte Verb konkretisieren. Die auf diese Weise ermittelten Systemelemente werden in einer Liste gespeichert.

Schritt 5) In diesem Schritt wird festgestellt, ob es ein Substantiv gibt, das als Input für die Funktion spezifiziert und vom Algorithmus noch nicht betrachtet wurde.

Wurde ein solches Substantiv gefunden wird mit Schritt 6 fortgefahren. Ansonsten wird Schritt 9 ausgeführt.

Schritt 6) In diesem Schritt wird für das jeweilige Input-Substantiv die zugehörige äquivalente Gruppe ermittelt.

Schritt 7) In diesem Schritt werden alle Substantivgruppen ermittelt die Substantive enthalten, die das gerade betrachtete Input-Substantiv konkretisieren.

Schritt 8) In diesem Schritt wird festgestellt, ob es ein Substantiv gibt, das als Output für die Funktion spezifiziert und vom Algorithmus noch nicht betrachtet wurde. Wurde ein solches Substantiv gefunden wird mit Schritt 9 fortgefahren. Ansonsten wird Schritt 12 ausgeführt.

Schritt 9) In diesem Schritt wird für das jeweilige Output-Substantiv die zugehörige äquivalente Substantivgruppe ermittelt.

Schritt 10) In diesem Schritt werden alle Substantivgruppen ermittelt die Substantive enthalten, die das gerade betrachtete Output-Substantiv konkretisieren.

Schritt 11) In diesem Schritt werden die in Schritt 6 und 7 ermittelten Substantivgruppen jedes einzelnen Input-Substantivs der betrachteten Funktion ermittelt. Gibt es ein Input-Substantiv, dann wird mit Schritt 12 fortgesetzt, ansonsten mit Schritt 14.

Schritt 12) In Schritt 12 werden die Input-Substantive der von den Systemelementen erfüllten Funktionen untersucht. Gehört eines dieser Input-Substantive zu den in Schritt 6 und 7 ermittelten Substantivgruppen, dann erfüllt das Systemelement dieses Input-Substantiv und es wird mit Schritt 11 fortgesetzt. Gehört keines der Input-Substantive zu den Substantivgruppen, dann erfüllt das Systemelement bezogen auf die gerade betrachtete Systemelementfunktion die Funktion nicht und es geht mit Schritt 13 weiter.

Schritt 13) Die Systemelementfunktion wird aus der in Schritt 4 erstellten Liste gestrichen.

Schritt 14) wie Schritt 11, nur mit Output-Substantiven.

Schritt 15) wie Schritt 12, nur mit Output-Substantiven.

Schritt 16) Wie Schritt 13.

Schritt 17) Hierbei handelt es sich wieder um einen Zustand. Die jetzt in der Liste noch vorhandenen Systemelemente erfüllen sowohl das spezifizierte Verb, als auch alle Input- und Output-Substantive. Diese können jetzt dem Entwickler präsentiert werden.

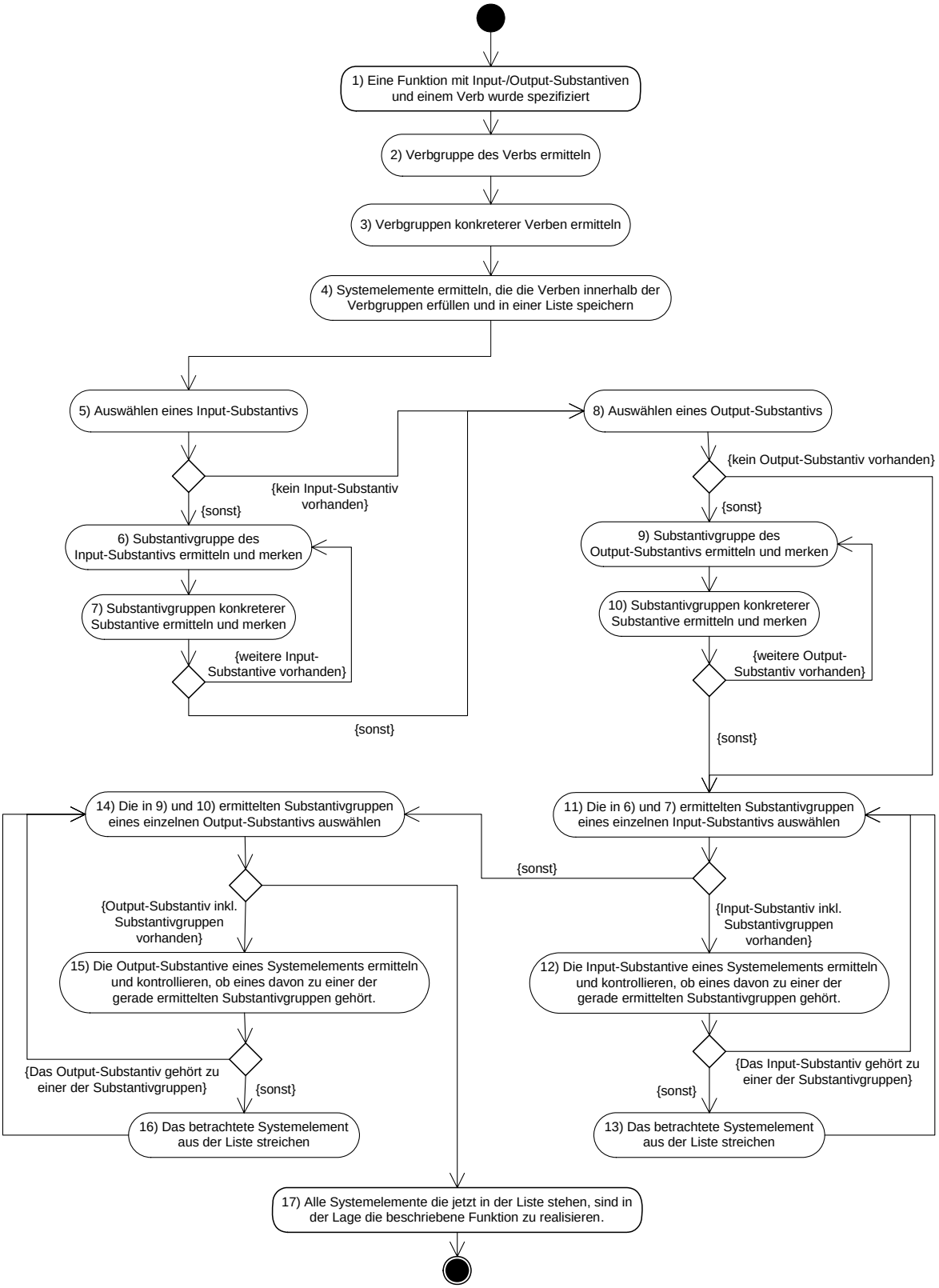


Abbildung 64: Ablauf der Systemelementsuche

7.5 Einsatz von Graphtransformationsregeln

Die formale Beschreibung des Suchalgorithmus erfolgt mit Hilfe von Graphtransformationsregeln. Graphtransformationsregeln wurden gewählt, um den Suchalgorithmus auf einer möglichst hohen Abstraktionsebene zu beschreiben. Dies hilft zum

einen dabei den Algorithmus besser zu verstehen und zum anderen ermöglichen es Graphtransformationsregeln den Algorithmus sehr einfach anzupassen. Dies kann z.B. erforderlich werden, wenn die Suchstrategie aufgrund von Optimierungsüberlegungen angepasst werden soll.

Die Beschreibung der Graphtransformationsregeln erfolgt mit Hilfe von *Story-Pattern.* Sie wurden 2002 von Zündorf [Zün02] entwickelt und basieren auf den Überlegungen und Ideen zur Einführung eines formalen Konsistenzbegriffs für Dokumente im Softwarelebenszyklus. Erste Untersuchungen und prototypische Realisierungen wurden hierzu bereits im IPSEN-Projekt [Nagl96] durchgeführt und von Zündorf entsprechend erweitert. Prinzipiell wären auch andere Graphtransformationsansätze, wie beispielsweise die *Design Graph Grammers* von Stein [Ste01], denkbar. Hierbei werden beim finden der Anwendungsstelle (Graph-Matching) allerdings zusätzliche Kontext-Informationen berücksichtigt. Diese Kontext-Informationen beinhalten mehr, als nur die Angaben bzgl. der linken Regelseite, was insbesondere zur Analyse und Synthese von Designdokumenten verwendet wird. Im Rahmen dieser Arbeit werden jedoch speziell Konsistenzsicherungsmechanismen (vgl. Kapitel 8) benötigt, bei denen zusätzliche Kontextinformationen eher kontraintuitiv wären, da die verwendete Semantik deutlich komplizierter ist. Der Einsatz von Story-Pattern zur Konsistenzsicherung hat sich sehr bewährt und werden daher durchgängig innerhalb dieser Arbeit verwendet.

Ein Story-Pattern besteht prinzipiell aus zwei Graphen, der linken und der rechten Regelseite. Die beiden Graphen müssen wohlgeformte, objektorientierte Graphen sein und sich auf das gleiche Modell (UML-Klassendiagramm) beziehen.

Die Ausführung eines Story-Patterns auf einem Graphen besteht aus der Suche nach einer Anwendungsstelle für die Regel, die durch die linke Regelseite definiert ist, und der anschließenden Ersetzung der Anwendungsstelle durch die rechte Regelseite. Die Wahl der Anwendungsstelle ist dabei nicht-deterministisch, was zu einer Menge möglicher Ergebnisgraphen führen kann.

Als Notation für Story-Pattern werden UML-Kollaborationsdiagramme verwendet, wobei die linke und rechte Regelseite in einem Diagramm dargestellt werden. Hinzuzufügende Elemente werden mit <<create>> und zu löschende Elemente mit <<destroy>> gekennzeichnet. Die Grenzen der Regel werden durch einen Rahmen dargestellt.

Story-Pattern können durch Kotrollflüsse miteinander verbunden werden. Mehrere, durch einen Kontrollfluss verbundene Story-Pattern werden als *Story-Diagramme* bezeichnet. Die Notation der Story-Diagramme beruht auf UML-Aktivitätendiagrammen, wobei ein Story-Pattern eine einzelne Aktivität des Story-Diagramms darstellt. Eine Methode einer Klasse wird als Story-Diagramm definiert, d.h. das Story-Diagramm beschreibt das Verhalten der Methode. Innerhalb eines Story-Diagramms existiert ein eigener Namensraum.

Bevor in Abschnitt 7.6.1 mit der genauen formalen Beschreibung des Algorithmus begonnen wird, soll in den nächsten Abschnitten kurz die Funktionsweise von Story-Diagrammen bzw. Story-Pattern erläutert werden.

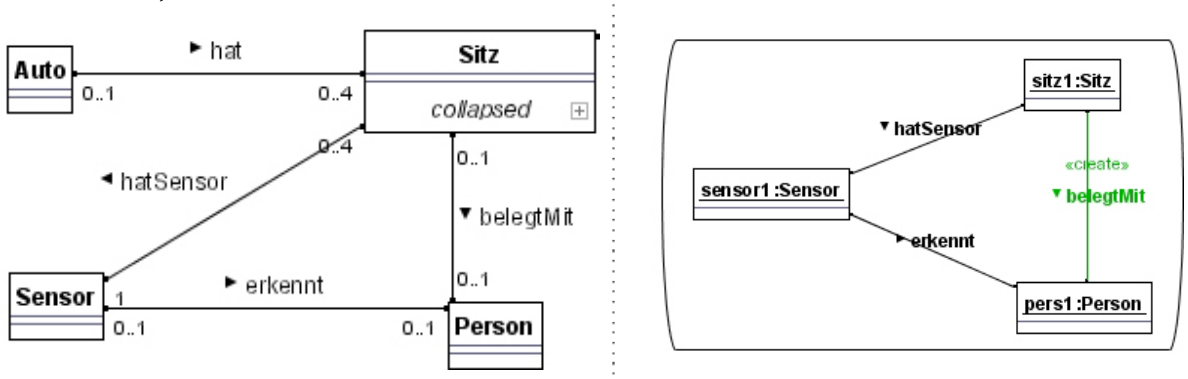
7.5.1 Funktionsweise von Story-Pattern

Jedes Story-Pattern basiert auf einem Typgraph, der in Form eines UML-Klassendiagramms dargestellt wird (vgl. linke Seite der Abbildung 65)[28]. Auf Basis dieses Klassendiagramms werden Objekte instanziiert und miteinander verlinkt. Der so entstehende Graph wird als

[28] Die Abbildungen der Story-Pattern, innerhalb dieser Arbeit, wurden in einem Werkzeug erstellt und von dort übernommen. Trotz der etwas geringen Qualität wurde darauf verzichtet sie nachzuzeichnen, um die durch das Werkzeug durchgeführte syntaktische Korrektheitsanalyse nicht zu kontrakarrieren.

Wirtsgraph bezeichnet und lässt sich mit Hilfe des Story-Patterns modifizieren (z.B. Erzeugen oder Löschen von Objekten oder von Links).

In dem Typgraphen aus Abbildung 65 hat ein Auto maximal vier Sitze. Auf jedem Sitz kann entweder keine oder genau eine Person sitzen. Des Weiteren ist ein Sensor ist in der Lage festzustellen, ob eine Person auf einem der Sitze sitzt oder nicht.



Typgraph StoryPattern

Abbildung 65: Beispiel für die Funktionsweise von Story-Pattern

Auf der rechten Seite von Abbildung 65 befindet sich ein Story-Pattern welches folgende Transformation vornimmt:

Sobald der Sensor feststellt, dass sich eine Person auf einem Sitz befindet, wird diese Person dem entsprechenden Sitz zugeordnet. Aus graphentechnischer Sicht bedeutet dies, sobald im Wirtsgraphen ein Link zwischen einem Sitz und einem Sensor (Assoziation *hatSensor*) und diesem Sensor und einer Person (Assoziation *erkennt*) gefunden wird, soll ein neuer Link (Assoziation *belegtMit*) zwischen dem Sitz und der Person erzeugt werden.

Dieses Beispiel gibt einen ersten Eindruck davon, wie Graphtransformationen mit Hilfe von Story-Pattern beschrieben werden. Für weiterführende Informationen sei auf die Arbeiten von A. Zündorf [Zün02] verwiesen.

7.5.2 Problem der Anwendungsstelle

Im Beispiel des letzten Abschnitts wurde gezeigt, wie Story-Pattern genutzt werden. Das Problem dabei ist jedoch, dass die Ausführung von Story-Pattern, also die Ausführung von Graphtransformationsregeln, das Graph-Isomorphie-Problem beinhaltet. Bezogen auf das obige Beispiel bedeutet dies, dass prinzipiell erst einmal alle möglichen Kombinationen der Objekte der Klassen Sitz, Sensor und Person überprüft werden müssten, bis ein entsprechender Teilgraph gefunden wurde der diese Objekte über Links der angegebenen Assoziationen verbindet. Diese Überprüfung aller Kombinationen stellt ein NP-Vollständiges Problem dar (vgl. [Meh84]).

Eine Möglichkeit, den Aufwand für die Teilgraphensuche zu verringern ist, die Anzahl der möglichen Anwendungsstellen, also die Anzahl der zu suchenden Teilgraphen, zu reduzieren. In unserem Beispiel würde dies bedeuten, dass nicht jede mögliche Kombination von Sitz, Sensor und Person überprüft werden muss, sondern z.B. nur die Kombinationen aller Sensoren und Personen mit genau einem, konkret festgelegten Sitz. Dies würde die Laufzeit erheblich verkürzen und führt im Durchschnitt zu polynomiellem oder sogar linearem Laufzeitverhalten.

Eine in der Praxis häufig eingesetzte Methode ist es den Teilgraphen durch eine Menge von Knoten und Kanten zu beschreiben, wobei die beschriebenen Knoten an Knoten des Wirtsgraphen gebunden werden. Knoten eines Story-Patterns, die bereits an Knoten des

Wirtsgraphen gebunden sind, werden explizit angegeben und werden als *gebundene Objekte* bezeichnet.

Gebundene Objekte werden ohne Angabe ihrer Klasse im Story-Pattern dargestellt. In Abbildung 66 ist der sitz1 ein gebundenes Objekt. Dies bedeutet, dass ausgehend von diesem einen Objekt lediglich noch die Kombinationen mit den Sensoren und Personen überprüft werden müssen. Es wird demnach nur noch ein Sitz überprüft und nicht alle.

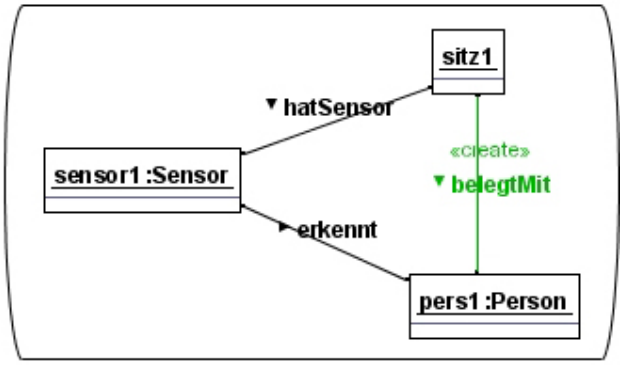


Abbildung 66: Story-Pattern Beispiel mit gebundenen Objekten

7.5.3 Festlegen des Anwendungskontextes

In Abschnitt 7.5.2 wurde gezeigt, wie das Laufzeitverhalten mit Hilfe gebundener Objekte reduziert werden kann. Es stellt sich allerdings noch die Frage, an welchen Knoten im Wirtsgraphen die Objekte gebunden werden, d.h. wie sieht der genaue Anwendungskontext aus? Bisher wurde nur gezeigt, was gebundene Objekte sind und wie sie modelliert werden. Es wurde noch nichts darüber gesagt, wie man ein im Story-Pattern gebundenes Objekt mit einem Knoten im Wirtsgraphen in Beziehung setzt.

Eine Möglichkeit den Anwendungskontext zu beschreiben ist, einzelne Knoten im Wirtsgraphen zu markieren. Dies kann z.B. durch spezielle Markierungsknoten erreicht werden. Sind diese Markierungsknoten eindeutig, so wird die Anwendungsstelle des Story-Patterns eingeschränkt.

Das Beispiel in Abbildung 67 zeigt das Story-Diagramm der Methode checkForPerson der Klasse Sitz. Der Methode wird das Objekt sensor1 übergeben. Einen Rückgabewert gibt es nicht (void). Der Kontrollfluss beginnt bei der Startaktivität, dargestellt durch einen ausgefüllten Kreis und endet bei der Stop-Aktivität, dargestellt durch einen doppelten Kreis. Das Story-Pattern oben links wird als erstes ausgeführt. Ist eine Anwendungsstelle gefunden und alle Modifikationen wurden durchgeführt, so wird die success-Kante verfolgt und das untere Story-Pattern wird ausgeführt. Im Falle eines Fehlschlags wird die failure-Kante verfolgt und das rechte Story-Pattern wird ausgeführt.

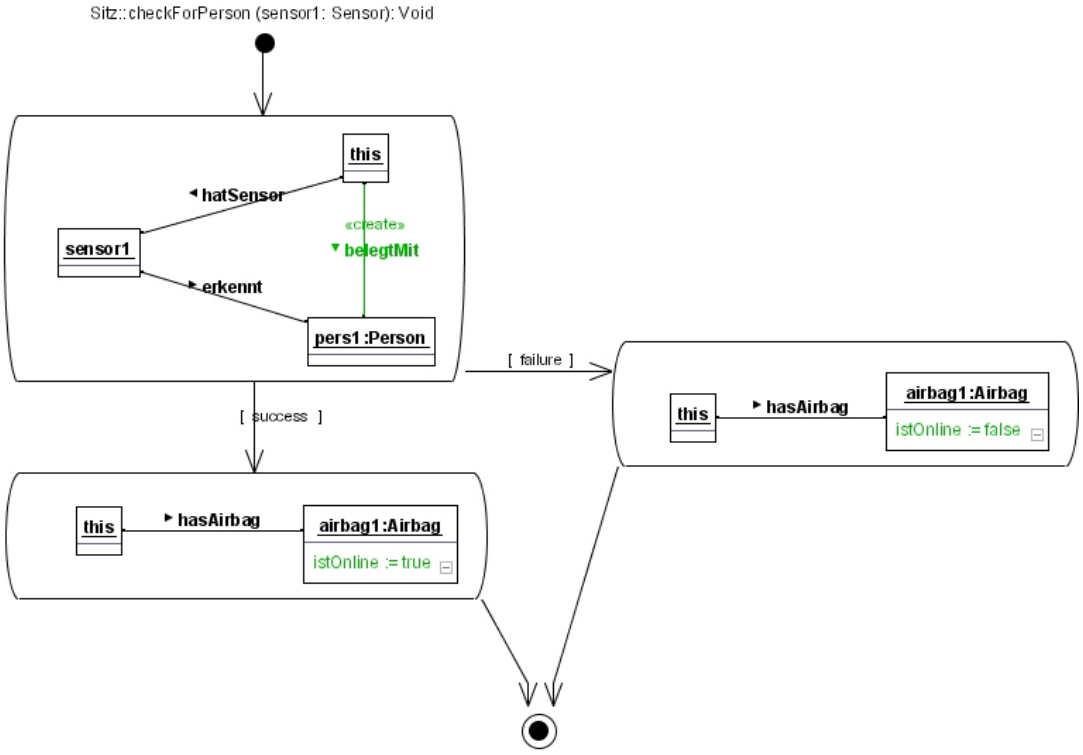


Abbildung 67: Story-Diagramm Beispiel

Das obere Story-Pattern beinhaltet zwei gebundene Objekte this und sensor1. Der Anwendungskontext für sensor1, also ein existierender Knoten im Wirtsgraphen, wird der Methode übergeben. Der Anwendungskontext für this ist das aktuelle Objekt, auf dem die Methode aufgerufen wurde.

Wurde die success-Kante verfolgt, dann wird im unteren Story-Pattern ein Objekt der Klasse Airbag gesucht, welches einen Link zum Objekt this hat. Wurde ein solcher Link gefunden, dann wird das Attribut istOnline auf true gesetzt. Wurde die failure-Kante verfolgt, dann wird das Attribut istOnline auf false gesetzt.

7.6 Formale Beschreibung des Suchalgorithmus

Im Folgenden werden nun die in Abschnitt 7.4 allgemein beschriebenen Schritte des Suchalgorithmus mit Hilfe der in 7.5 beschriebenen Story-Diagrammen bzw. Story-Pattern formal spezifiziert. Die Grundlage bildet dabei ein einheitliches Modell in Form eines UML-Klassendiagramms. Darauf aufbauend werden dann die einzelnen Schritte durch ein oder mehrere Story-Diagramme formalisiert.

7.6.1 Datenmodell

Der Typgraph, auf dem die StoryPattern bzw. StoryDiagramme spezifiziert werden ist in Abbildung 63 in Form eines UML-Klassendiagramms dargestellt.

7.6.2 Der Suchalgorithmus

Nachdem der Entwickler eine erste Funktionshierarchie aufgestellt hat, hat er die Möglichkeit für einzelne Funktionen dieser Funktionshierarchie Systemelemente zu ermitteln. Hierzu wird ihm die Methode startSearch() der Klasse Function zur Verfügung gestellt.

Im ersten Story-Pattern (1) in Abbildung 68 wird das Verb (vn1) ermittelt, das der Funktion (this) zugewiesen wurde. Daraufhin wird ein Objekt (sr1) der Klasse SearchResult erzeugt und sowohl mit der Funktion (this), als auch mit der Gruppe äquivalenter Verben (vg1) verlinkt.

Als nächstes wird nun für die zuvor identifizierte äquivalente Gruppe von Verben ermittelt, welche anderen äquivalenten Gruppen existieren, die diese Gruppe konkretisieren (vgl. Schritt 1 und 2 aus 7.4). Zur Ermittlung der konkreteren Gruppen äquivalenter Verben wurde in Story-Pattern (2) ein rekursiver Algorithmus spezifiziert. Dieser rekursive Algorithmus ist in der Methode checkChildren(sr1) spezifiziert. Nach Beendigung der Methode und damit der Rekursion, sind alle Gruppen äquivalenter Verben ermittelt, die die in Story-Pattern (1) ermittelte Gruppe konkretisieren. Eine detaillierte Erläuterung der Methode checkChildren(sr1) erfolgt in Abschnitt 7.6.3.

Nachdem alle äquivalenten Gruppen von Verben ermittelt wurden, werden nun im nächsten Story-Pattern (3) alle Funktionen der Systemelemente ermittelt (Klasse SE_Function), die Verben erfüllen, die zu diesen Gruppen gehören (vgl. Schritt 3 aus 7.4). Die gefundenen SE_Functions (sef1) werden als potentielle Lösungen mit dem Objekt sr1 verlinkt. Über die realizedBy-Assoziation lassen sich nun alle SE_Function und damit alle Systemelemente (Assoziation: functions) ermitteln, die die Funktion prinzipiell erfüllen. Hierbei sind allerdings die Input- und Output-Substantive noch nicht berücksichtigt. Dies erfolgt im nächsten Story-Pattern (4).

Mit Hilfe der Methode checkForInputSubjects(RequiredSubjects) der Klasse Function werden für jedes Input-Substantiv die zugehörige äquivalente Gruppe ermittelt. Für diese äquivalente Gruppe werden dann diejenigen Gruppen ermittelt, die diese Gruppe konkretisieren (vgl. Schritt 4, 5 und 6 aus 7.4). Die Ermittlung der konkreteren Gruppen erfolgt ebenfalls mit Hilfe einer Rekursion. Das gleiche Vorgehen wird das mit Hilfe der Methode checkForOutputSubjects(RequiredSubjects) für die Output-Substantiven wiederholt (vgl. Schritt 7, 8, 9 aus 7.4). Eine genaue Beschreibung der Methoden erfolgt in Abschnitt 7.6.5.

Als letzten Schritt müssen nun noch diejenigen SE_Functions ermittelt werden, die die geforderten Input- bzw. Output-Substantive NICHT erfüllen (vgl. Schritt 10 aus 7.4). Diese müssen aus der Menge der potentiellen Lösungen entfernt werden. Dies erfolgt mit Hilfe der Methoden removeWrongElementsInput() und removeWrongElementsOutput(). Die Beschreibung dieser Methoden befindet sich in Abschnitt 7.6.7.

Nachdem nun alle Objekte der Klasse SE_Function entfernt wurden, die die geforderten Input- bzw. Output-Substantive nicht erfüllen, sind über die Assoziation realizedBy nur noch die Objekte zu erreichen, die die spezifizierte Funktion realisieren. Über die Assoziation fulfills können dann die zugehörigen Systemelemente ermittelt und dem Entwickler präsentiert werden.

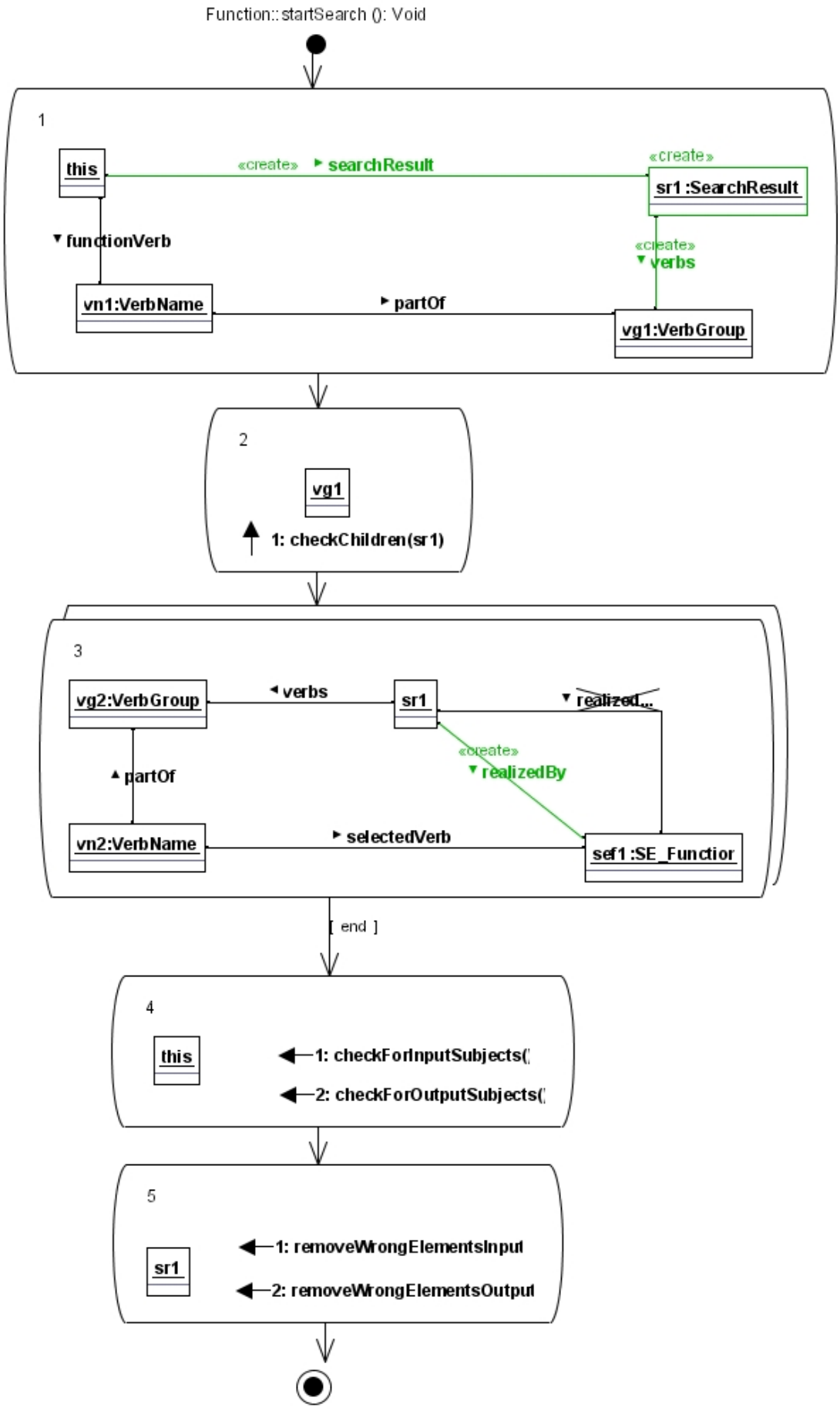


Abbildung 68: Story-Diagramm des Suchalgorithmus

7.6.3 Konkretisierungsrelation der Verben

Mit Hilfe der Methode checkChildren(SearchResult) werden diejenigen Gruppen von Verben ermittelt, die die aktuelle Gruppe konkretisieren.

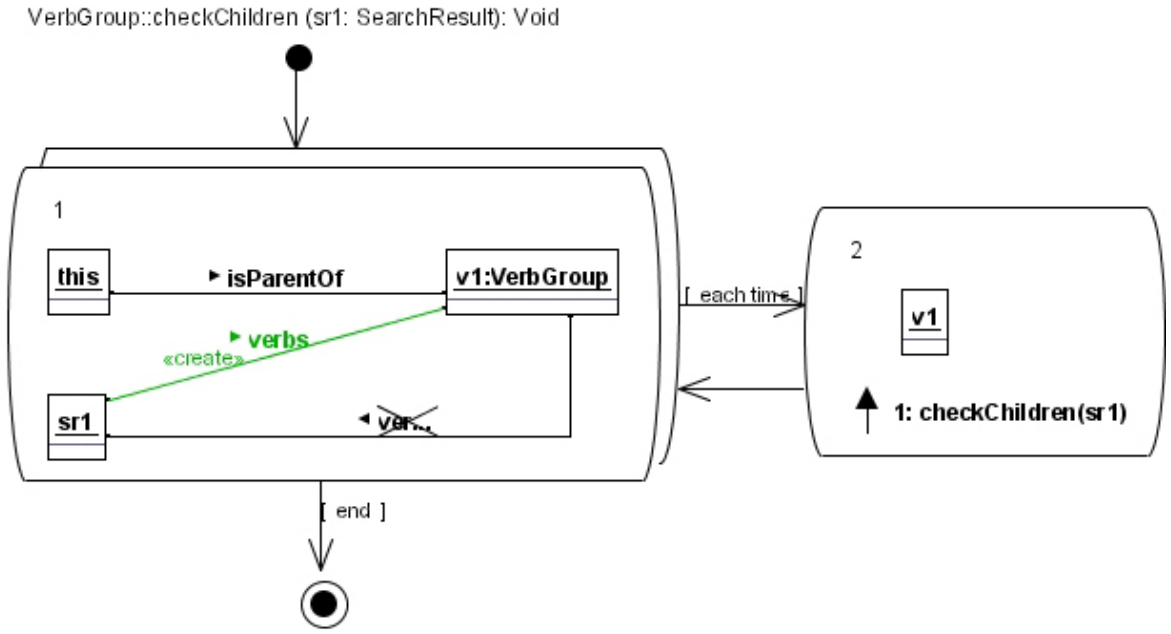


Abbildung 69: Konkretere Verben ermitteln

Im ersten Story-Pattern (1) werden alle Gruppen (vg1) ermittelt, die durch einen Link der Assoziation isParentOf mit der aktuellen Gruppe (this) verbunden sind. Für jede dieser Gruppen wird ein Link zum übergebenen SearchResult-Objekt (sr1) erstellt, wenn ein solcher Link nicht bereits existiert.

Bevor eine weitere Gruppe ermittelt wird, wird mit Story-Pattern 2 fortgesetzt. In diesem Story-Pattern wird auf der gerade ermittelten Gruppe (vg1) die Methode checkChildren(SearchResult) erneut aufgerufen.

Sobald keine Gruppen (Objekte der Klasse VerbGroup) über die Assoziation isParentOf mehr erreicht werden können, ist die Methode beendet. Jetzt sind mit dem SearchResult-Objekt (sr1) alle Gruppen verlinkt (Assoziation: verbs).

Tiefensuche

Aus graphentechnischer Sicht handelt es sich bei der Methode checkChildren(SearchResult) um eine Tiefensuche. Mit den Assoziationen isParentOf wird ein Graph aufgespannt und rekursiv durchlaufen.
Ausgehend von einer Gruppe äquivalenter Verben (Klasse: VerbGroup) werden alle Gruppen ermittelt, die eine Konkretisierung darstellen (über die Assoziation isParentOf). Für diese konkreteren Gruppen wird dann wiederum nach Gruppen gesucht die ihrerseits konkreter sind als die vorhergehende Gruppe. Dies wird so lange fortgesetzt, bis für eine Gruppe keine konkreteren Gruppen vorhanden sind. Dann springt der Algorithmus zurück. Der Algorithmus endet, wenn alle konkreteren Gruppen ermittelt wurden.
In Abbildung 70 ist das Vorgehen beispielhaft skizziert. Begonnen wird bei der äquivalenten Gruppe mit den Verben wandeln, umwandeln und verwandeln (1). Dann wird der Link isParentOf zur Gruppe mit dem Verb erzeugen verfolgt (2). Danach erfolgt der Rücksprung zur ersten Gruppe und der Sprung zur Gruppe mit dem Verb verbrennen (3). Von hier geht es weiter zur Gruppe mit den Verben zerstören und vernichten (4). Von hier erfolgt der

Rücksprung zur Gruppe mit dem Verb verbrennen und von dort zurück zur ersten Gruppe. Als nächstes erfolgt der Sprung zur Gruppe mit dem Verb messen (5). Von dort geht es wieder zurück und weiter zur Gruppe mit dem Verb absorbieren (6). Nun geht es nochmals zurück und weiter zur Gruppe mit dem Verb zersetzen (7). Von hieraus existiert zwar ein Link zur Gruppe mit den Verben zerstören und vernichten, da diese Gruppe aber bereits besucht wurde, erfolgt kein Sprung zu dieser Gruppe. Schließlich erfolgt noch der Rücksprung zur ersten Gruppe. Da es jetzt keine weiteren Links mehr gibt, bricht der Algorithmus ab.

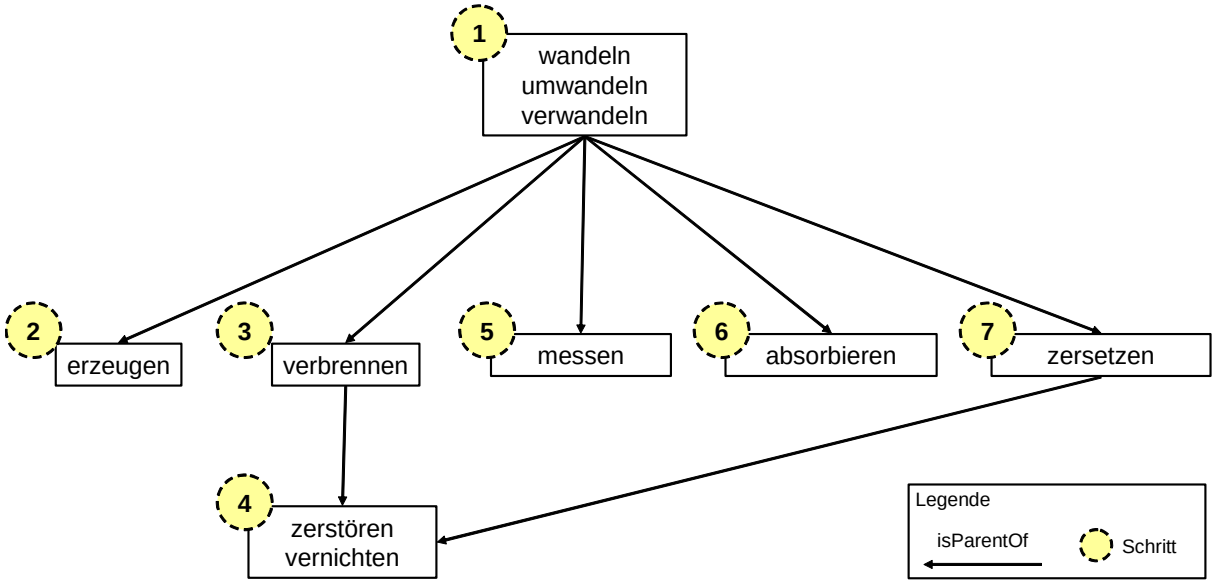


Abbildung 70: Vorgehen bei der Methode checkChildren(SearchResult).

7.6.4 Äquivalenzrelation der Verben

Die Berücksichtigung der Äquivalenzrelationen erfolgt implizit durch die gegebene Objektstruktur. Jedes Verb (Objekt der Klasse VerbName) gehört zu genau einer Gruppe äquivalenter Verben (Objekt der Klasse VerbGroup). Da die Suche auf Basis der Gruppe und nicht auf Basis der einzelnen Verben erfolgt, wird die Äquivalenz der Verben indirekt mit berücksichtigt.

7.6.5 Input-Substantive

Mit Hilfe der Methode checkForInputSubjects() werden zu jedem Input-Substantiv die entsprechende äquivalente Gruppe ermittelt und für diese Gruppe alle weiteren Gruppen ermittelt, die eine Konkretisierung darstellen. Dieses Vorgehen ähnelt dem in 7.6.3 und 7.6.4 vorgestellten Vorgehen mit dem Unterschied, dass hier mehrere Substantive zu berücksichtigen sind und nicht nur genau eins.

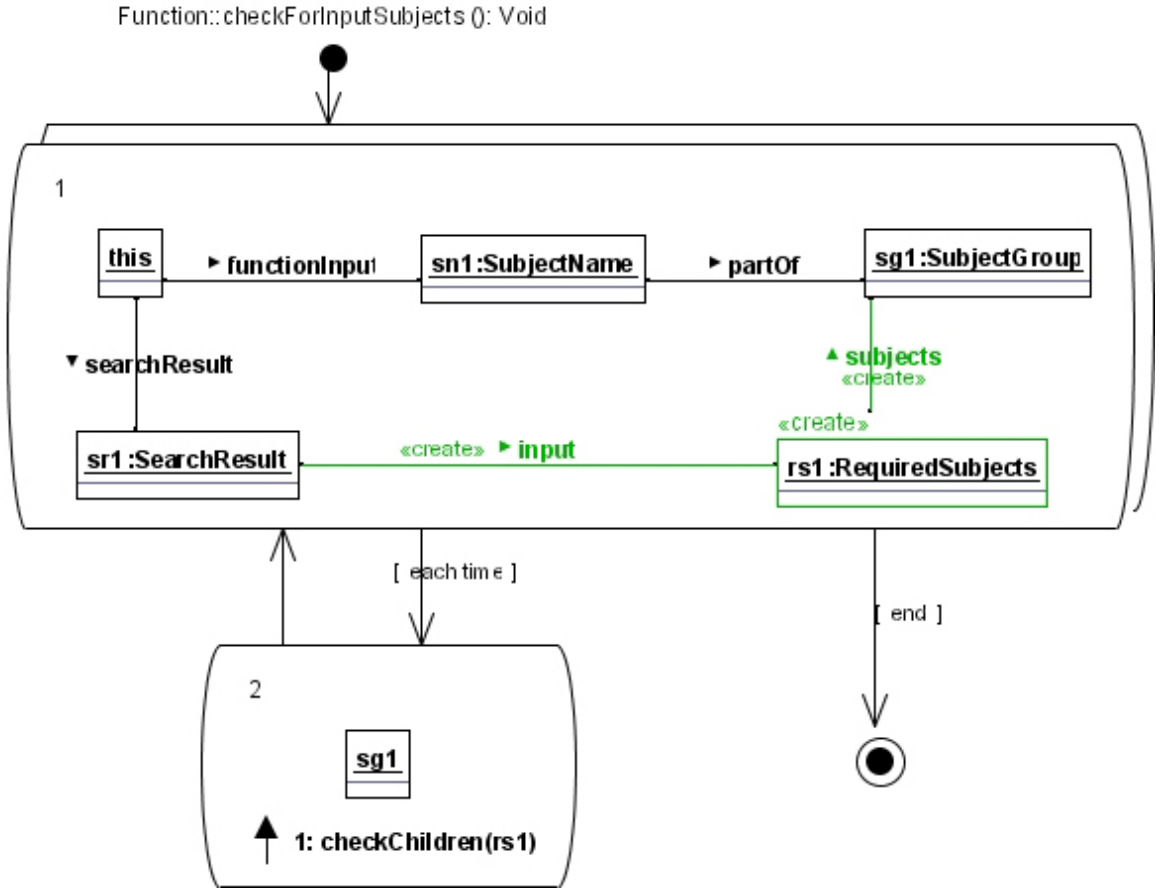


Abbildung 71: Input-Substantive berücksichtigen

Im ersten Story-Pattern (1) wird für jedes Input-Substantiv (sn1) des aktuellen Objekts (this) die zugehörige Gruppe (sg1) ermittelt. Für diese Gruppe wird dann ein Objekt der Klasse RequiredSubjects (rs1) angelegt und mit der Gruppe (sg1) und dem Objekt der Klasse SearchResult (sr1) verlinkt. Da jedes Substantiv zu genau einer Gruppe gehört, existieren nach Beendigung des Story-Pattern genau so viele Objekte der Klasse RequiredSubjects wie es Objekte der Klasse SubjectName gibt, die über die Assoziation functionInput erreicht werden können.

Mit Hilfe des ersten Story-Pattern werden allerdings nur diejenigen Gruppen ermittelt zu denen diejenigen Substantive gehören, die mit dem aktuellen Objekt (this) verlinkt sind. Alle Gruppen, die eine Konkretisierung darstellen werden so nicht erreicht. Da dies aber notwendig ist, werden für jede ermittelte Gruppe (sg1) die Gruppen ermittelt, die diese Gruppe konkretisieren (Story-Pattern (2)). Dort wird die Methode checkChildren(RequiredSubjects) aufgerufen. Diese Methode ermittelt rekursiv alle Gruppen mit konkreteren Substantiven. Die genaue Beschreibung dieser Methode befindet sich in Abschnitt 7.6.6.

Nachdem beide Story-Pattern abgearbeitet wurden, wurde für jedes Input-Substantiv ermittelt zu welcher Gruppe es gehört bzw. welche Gruppen eine Konkretisierung darstellen.

7.6.6 Konkretisierungsrelation der Substantive

Die Ermittlung der Gruppen von Substantiven die eine Konkretisierung einer anderen Gruppe von Substantiven darstellt, gleicht dem Vorgehen das in Abschnitt 7.6.3 vorgestellt wurde.

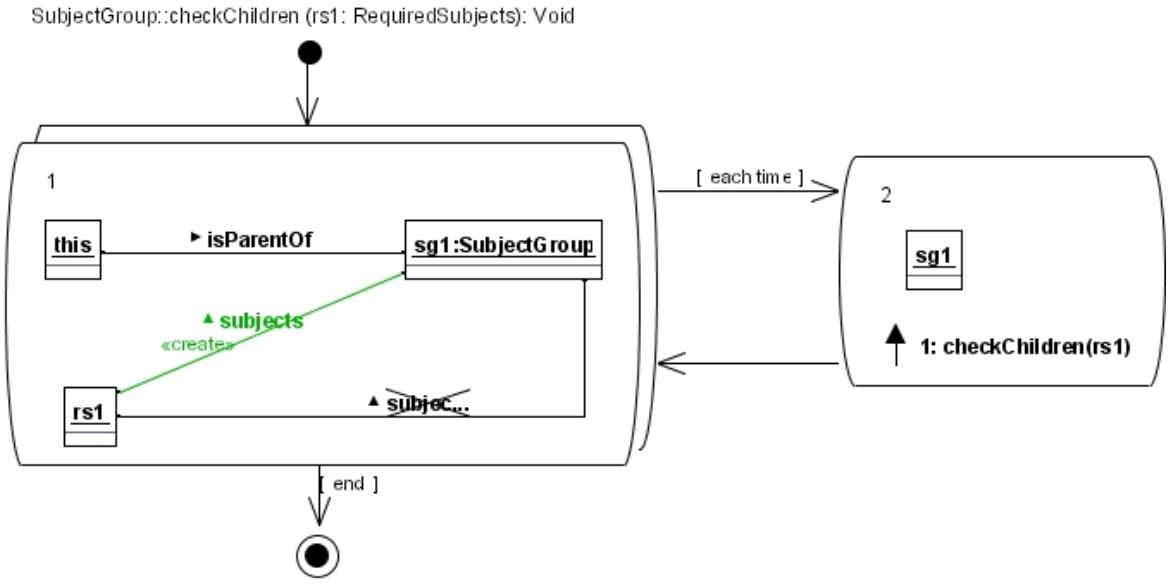


Abbildung 72: Ermittlung konkreter Gruppen

Im ersten Story-Pattern (1) werden alle Gruppen (sg1) ermittelt, die durch einen Link der Assoziation isParentOf mit der aktuellen Gruppe (this) verbunden sind. Für jede dieser Gruppen wird ein Link zum übergebenen SearchResult-Objekt (sr1) erstellt, wenn ein solcher Link nicht bereits existiert.

Bevor eine weitere Gruppe ermittelt wird, wird mit Story-Pattern 2 fortgesetzt. In diesem Story-Pattern wird auf der gerade ermittelten Gruppe (sg1) die Methode checkChildren(RequiredSubjects) erneut aufgerufen.

Sobald keine Gruppen (Objekte der Klasse SubjectGroup) über die Assoziation isParentOf mehr erreicht werden können, ist die Methode beendet. Jetzt sind mit dem SearchResult-Objekt (sr1) alle gefundenen Gruppen verlinkt (Assoziation: subjects).

7.6.7 Output-Substantive

Mit Hilfe der Methode checkForOutputSubjects(sr1) werden zu jedem Output-Substantiv die entsprechende äquivalente Gruppe ermittelt und für diese Gruppe alle weiteren Gruppen ermittelt, die eine Konkretisierung darstellen. Dieses Vorgehen ähnelt dem in 7.6.5 vorgestellten Vorgehen mit dem Unterschied, dass hier die Assoziationen functionInput und input durch die Assoziation functionOutput und output ersetzt wurden. Alles andere ist exakt das Gleiche und soll daher an dieser Stelle mehr im Detail erläutert werden.

Durch die Verwendung der äquivalenten Gruppen, sowohl für die Verben, als auch für die Substantive wird die unter 7.2 festgelegte Anforderung (4), „Die Äquivalenzrelation bei Substantiven und Verben müssen berücksichtigt werden“ erfüllt.

Des Weiteren wird durch die Berücksichtigung von Konkretisierungsbeziehungen die unter 7.2 festgelegte Anforderung (5), „Die Konkretisierungsrelation bei Substantiven und Verben müssen berücksichtigt werden“ erfüllt.

7.6.8 Reduzierung der möglichen Lösungen

Nach Beendigung der Story-Pattern eins bis vier der Methode startSearch(VerbName) (vgl. Abbildung 68) existiert genau ein Objekt der Klasse SearchResult und so viele Objekte der Klasse RequiredObjects wie es Input- und Output-Substantive gibt. Zusätzlich sind über die Assoziation realizedBy alle Objekte der Klasse SE_Function zu erreichen, die das spezifizierte Verb erfüllen.

Dies bedeutet, dass jetzt aus dieser Menge von Objekten der Klasse SE_Function diejenigen herausgefunden werden müssen, die die Input- bzw. Output-Substantive nicht erfüllen. Die zu diesen Objekten existierenden Links der Assoziation realizedBy müssen gelöscht werden.

Input-Substantive

Die Durchführung dieser Korrektur erfolgt in Story-Pattern 5 der Methode startSearch(VerbName). In diesem Story-Pattern wird zuerst die Methode removeWrongElementsInput() und dann die Methode removeWrongElementsOutput() aufgerufen.

Im ersten Story-Pattern (1) wird eine Schleife spezifiziert, die über alle Objekte der Klasse RequiredSubjects iteriert. Zur Erinnerung: Es gibt genau so viele Objekte wie es Input-Substantive gibt.

Für jedes dieser Objekte (rs1) wird eine weitere Schleife durchlaufen (2). In dieser Schleife werden alle Objekte der Klasse SE_Function ermittelt, die über einen Link der Assoziation realizedBy erreichbar sind. Zur Erinnerung: Das sind all diejenigen, die das in der Funktion spezifizierte Verb realisieren.

Im nächsten Story-Pattern (3) werden in einer Schleife alle über die Assoziation subjects erreichbaren Objekte der Klasse SubjectGroup ermittelt. Zur Erinnerung: Dies ist diejenige, zu denen ein Input-Substantiv gehört bzw. diejenigen die diese Gruppe konkretisieren.

Im anschließenden Story-Pattern (4) wird geprüft, ob ein Objekt der Klasse SubjectName (sn1) existiert, dass sowohl einen Link zum aktuellen Objekt der Klasse SubjectGroup (sg1) hat, als auch zum aktuellen Objekt der Klasse SE_Funktion (sef1). Ist dies der Fall, dann bedeutet es, dass das Systemelement, welches mit dem Objekt sef1 verlinkt ist, das untersuchte Input-Substantiv oder ein anderes Substantiv aus der gleichen Gruppe oder eines aus einer konkreteren Gruppe, erfüllt. Dies bedeutet das Objekt sef1 bleibt in der Menge der Lösungen enthalten und es kann mit dem nächsten fortgesetzt werden.

Kommt es jedoch vor, dass es kein Objekt der Klasse SubjectName (sn1) gibt, dass sowohl einen Link zum Objekt sg1 und einen Link zum Objekt sef1 hat, dann muss das Objekt sef1 aus der Liste der möglichen Lösungen entfernt werden.

Im fünften Story-Pattern wird der Link zwischen dem aktuellen Objekt (this) und dem Objekt der Klasse SE_Function gelöscht und mit dem nächsten Objekt der Klasse SE_Function fortgesetzt.

Nach Abschluss der Methode wurden alle Links zu Objekten der Klasse SE_Funktion gelöscht, die die geforderten Input-Substantive nicht erfüllen.

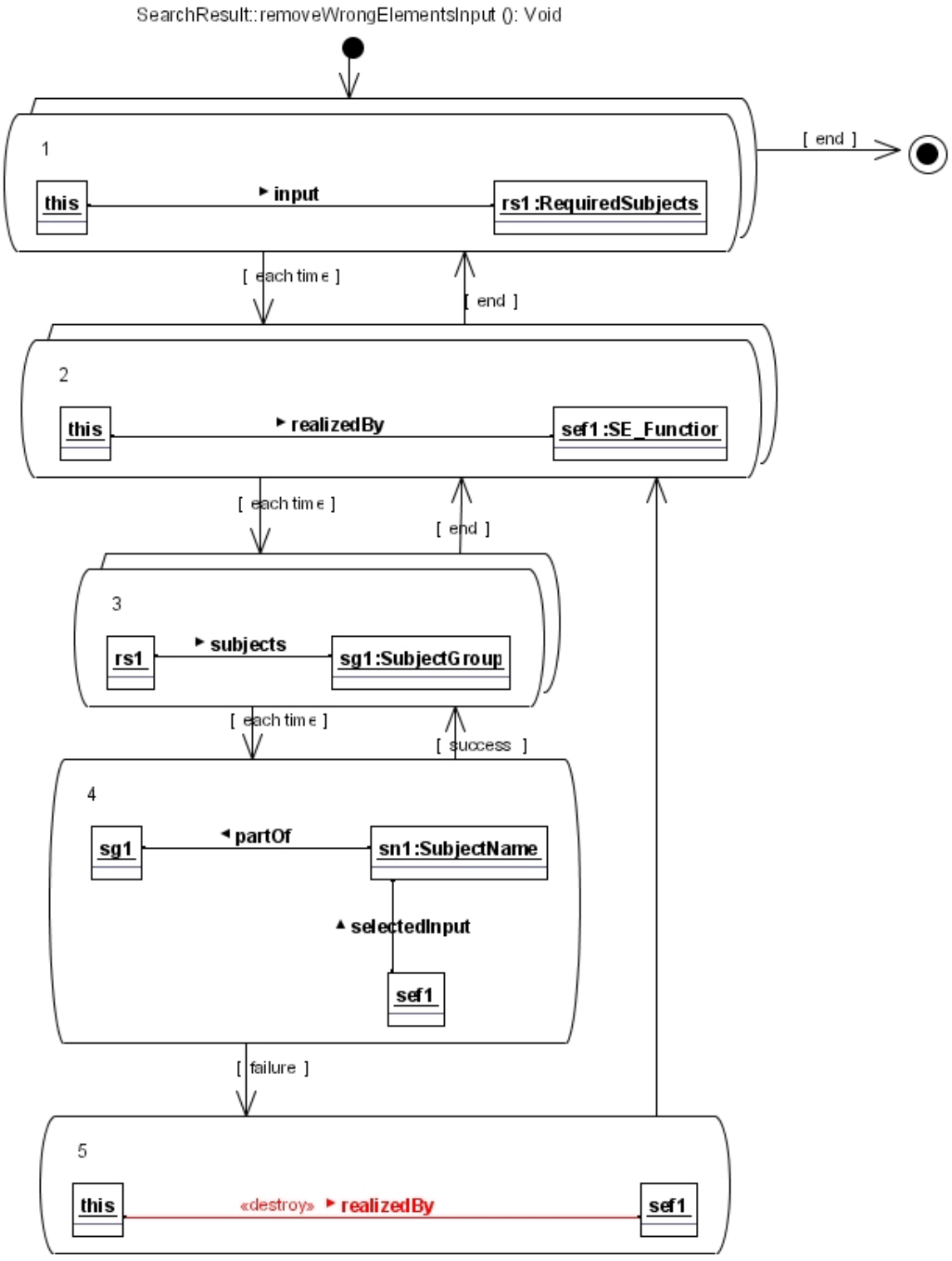


Abbildung 73: Filtern der Lösungen

Output-Substantive

Nachdem die Input-Substantive untersucht wurden, muss dies noch mit den Output-Substantiven erfolgen. Hierbei handelt es sich um die gleiche Vorgehensweise wie bei den Input-Substantiven, weshalb auf eine detaillierte Erklärung verzichtet wird.

Ergebnis

Nachdem alle Links der Assoziation realizedBy zu den Objekten der Klasse SE_Function gelöscht wurden, die die Output-Substantive nicht erfüllen, existieren nur noch Links der Assoziation realizedBy zu Objekten der Klasse SE_Function die als potentielle Lösungen in Frage kommen. Konkret bedeutet dies, dass mit jedem Objekt der Klasse SE_Function ein Objekt der Klasse Systemelement verlinkt ist, welches sowohl das geforderte Verb, als auch alle Input- und Output-Substantive erfüllt. Diese Systemelemente können nun dem Entwickler präsentiert werden.

7.6.9 Beispiel

Im Folgenden wird anhand eines Beispiels die Funktionsweise des Algorithmus verdeutlicht. In der Ausgangssituation existiert eine Systemelementbibliothek mit zwei Systemelementen. Für jedes der Systemelemente ist angegeben, welche Funktionen es erfüllt. Des Weiteren wurde eine Funktion, als Teilfunktion einer Funktionshierarchie definiert, für die nun ein Systemelement ermittelt werden soll.

Ausgangssituation

Im ersten Schritt werden alle, dem System bekannten, Substantive und Verben und deren Gruppen instanziiert. In Abbildung 74 sind es sieben Objekte der Klasse SubjectName (Ovale), vier Objekte der Klasse VerbName (Ovale), zwei Objekt der Klasse VerbGroup (Viereck mit abgerundeten Ecken) und fünf Objekt der Klasse SubjectGroup (Viereck mit abgerundeten Ecken). Zusätzlich werden die Objekte sef1, sef2 der Klasse SE_Function (Viereck) und die Objekte elektromotor und drehsensor der Klasse Systemelement (Sechseck) erzeugt. Alle erzeugten Objekte werden gemäß ihrer Assoziationen miteinander verlinkt. Die so entstehende Objektstruktur (Wirtsgraph) stellt die Basis für die Suche dar.

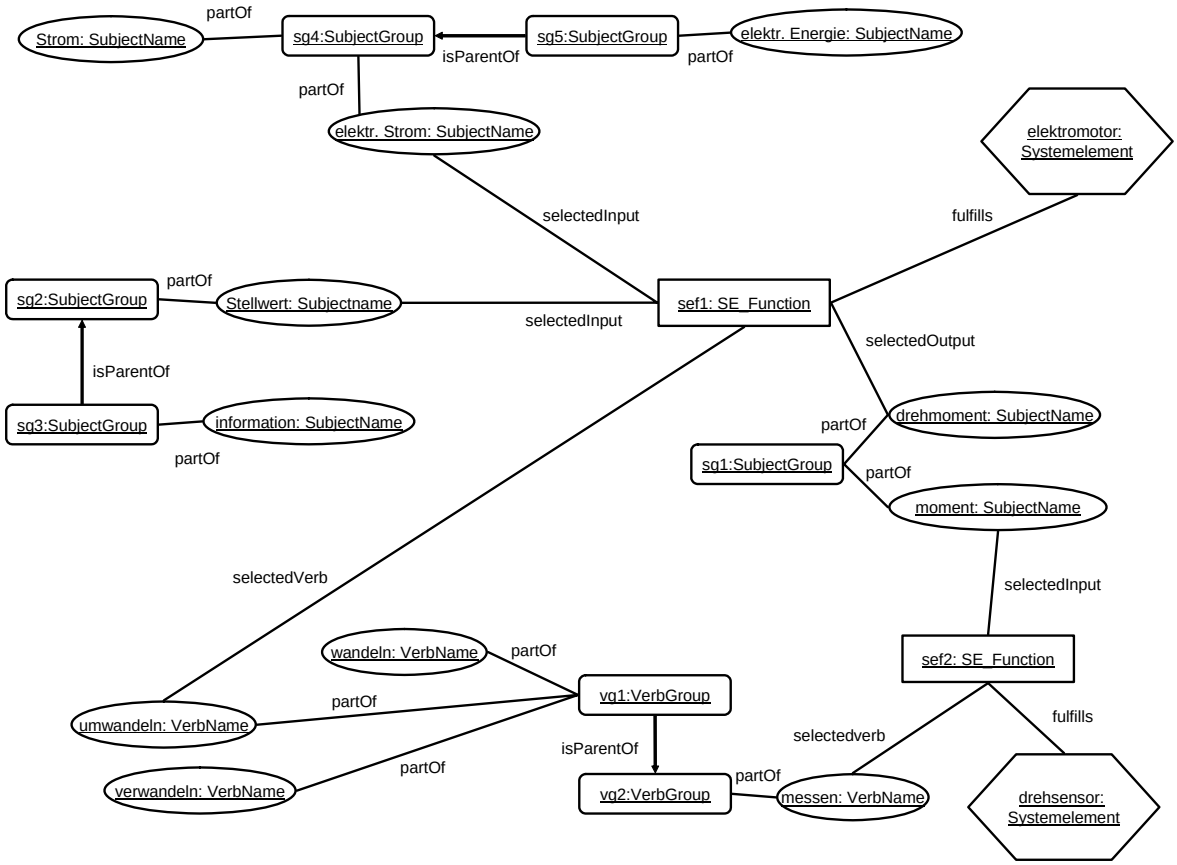


Abbildung 74: Ausgangssituation[29] (Objektdiagramm)

[29] Die verschiedenen Symbole dienen lediglich dazu um die Objekte einfacher zuordnen zu können.

Funktionsdefinition

Als nächstes soll nun für eine Funktion ermittelt werden, durch welche Systemelemente sie realisiert werden kann. Diese Funktion (vgl. Abbildung 75) hat als Input-Werte die Substantive elektr. Strom und information, als Verb das Wort wandeln und als Output-Wert das Substantiv moment.

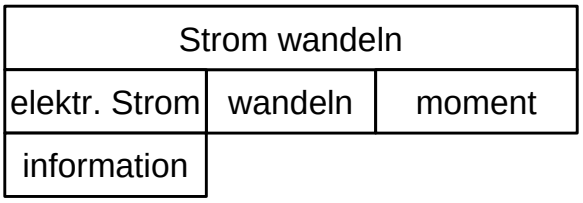


Abbildung 75: Funktionsbeschreibung

Bezogen auf die obige Objektstruktur bedeutet dies, dass ein neues Objekt f1 der Klasse Function erzeugt wird und mit den Objekten elektr. Strom, information und moment der Klasse SubjectName und dem Objekt wandeln der Klasse VerbName verlinkt wird (vgl. Abbildung 76).

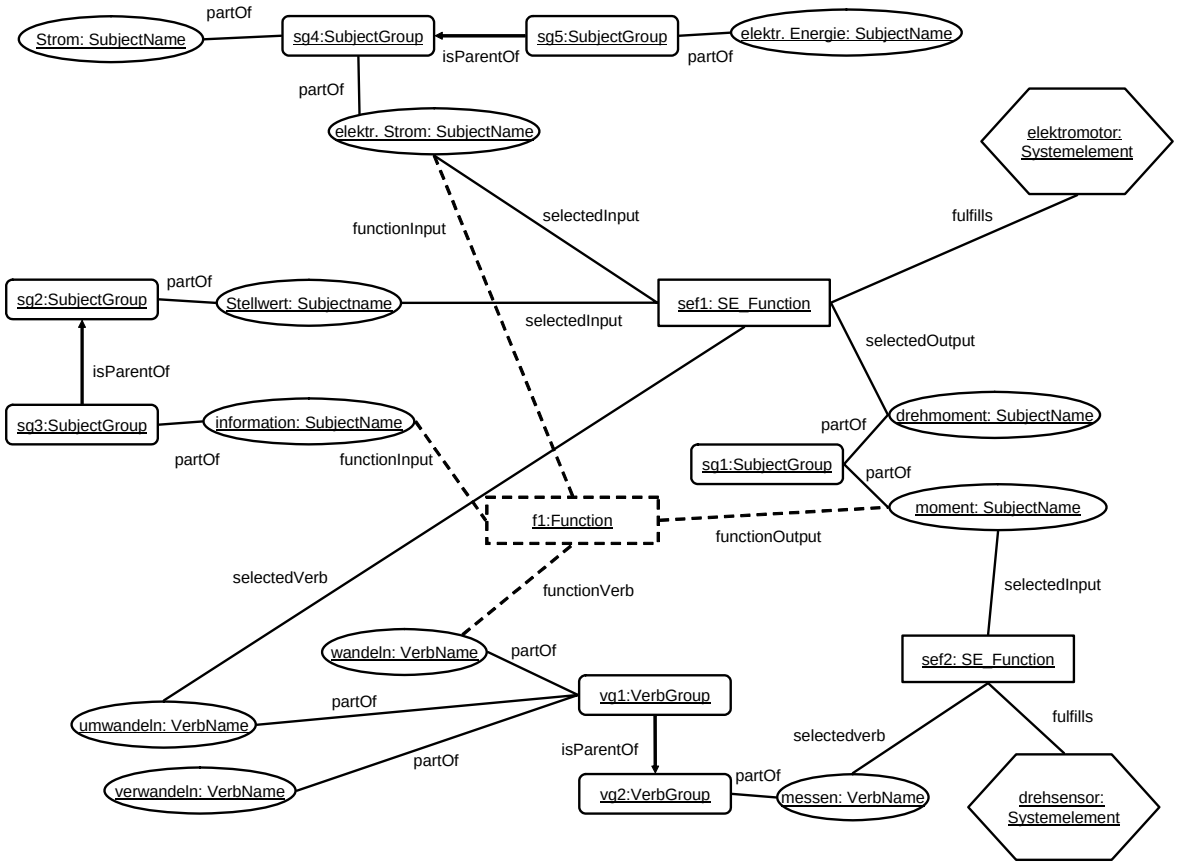


Abbildung 76: Objektstruktur inkl. Funktion

Hinweis: In den folgenden Abbildungen wurde aus Übersichtlichkeitsgründen teilweise auf die Anzeige von irrelevanten Assoziationsnamen verzichtet. Darüber hinaus werden die in der Abbildung neu erstellten Objekte oder Links gestrichelt dargestellt.

Ziel des Algorithmus ist es zu überprüfen, ob die Objekte die über die functionInput, functionOutput und functionVerb Links vom Objekt f1 aus erreichbar sind auch von den Objekten sef1 und sef2 erreicht werden. Auf den ersten Blick ist dies nicht der Fall, da z.B. vom Objekt sef1 über den Link functionInput das Objekt stellwert erreicht wird, nicht aber

das Objekt information. An dieser Stelle muss der Algorithmus die Links isParentOf und equals berücksichtigen, d.h. Objekte die über diese Links erreicht werden können, sind ebenfalls zulässig.

Ergebnis: Der Algorithmus sollte am Ende zu der Lösung kommen, dass das Objekt sef1 die Funktion f1 erfüllt, das Objekt sef2 jedoch nicht.

Berücksichtigung der Verben

Im ersten Story-Pattern (1) des Story-Diagramms in Abbildung 68 beginnt der Algorithmus damit ein Objekt der Klasse SearchResult anzulegen (vgl. Objekt sr1 in Abbildung 77). Dieses Objekt wird mit dem Objekt f1 und dem Objekt vg1 der Klasse VerbGroup verlinkt. Die Verlinkung mit dem Objekt vg1 erfolgt, da das gesuchte Verb wandeln zu dieser Gruppe gehört.

Im zweiten Story-Pattern des Story-Diagramms in Abbildung 68 wird zu dem Objekt vg1 das Objekt vg2 ermittelt, da dieses Objekt eine konkretere Gruppe von Verben darstellt. Dieses Objekt wird ebenfalls mit dem Objekt sr1 verlinkt.

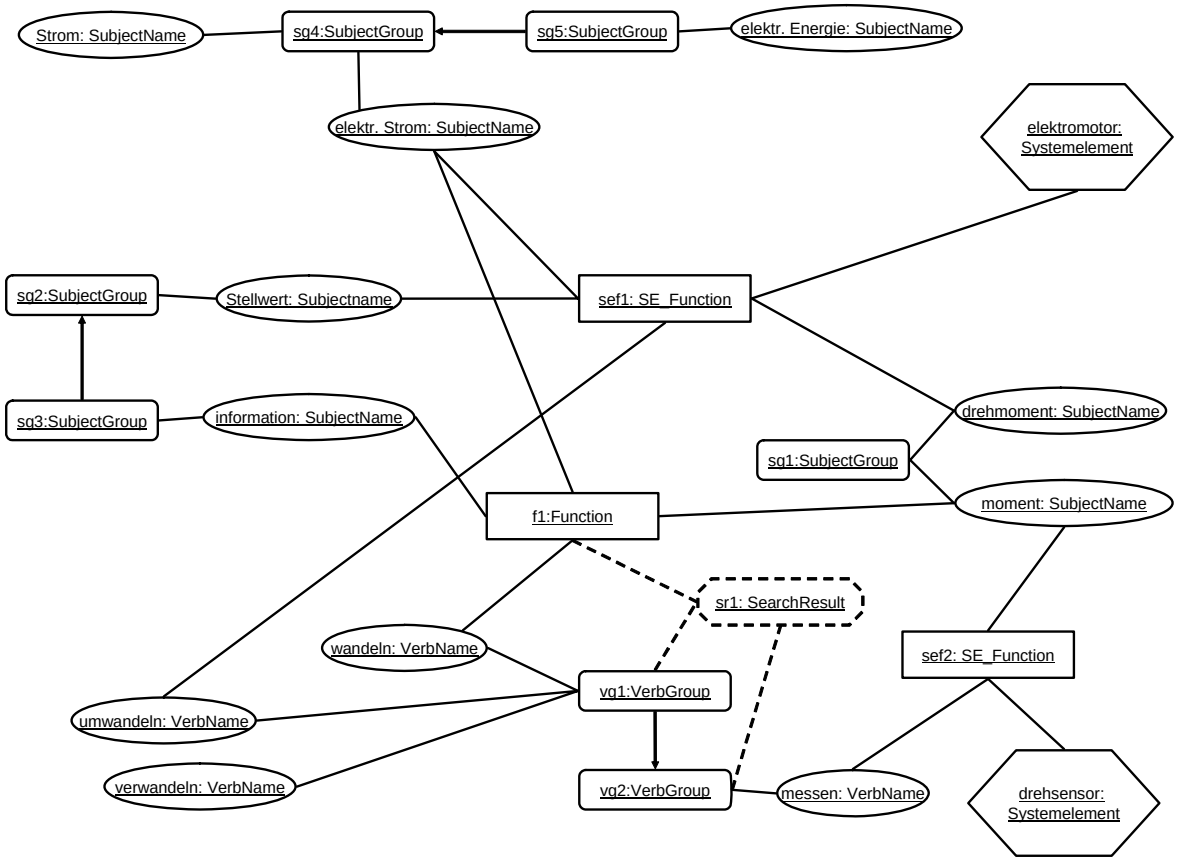


Abbildung 77: Verben

Ermitteln möglicher Systemelemente

In Story-Pattern drei des Story-Diagramms in Abbildung 68 werden die beiden Objekte der Klasse SE_Function (sef1 und sef2) mit dem Objekt sr1 verlinkt (vgl. Abbildung 78). Dies geschieht, da sie mit einem Objekt der Klasse VerbName verlinkt sind die selbst wiederum einen Link zu den ermittelten Objekten der Klasse VerbGroup (vg1 und vg2) haben.

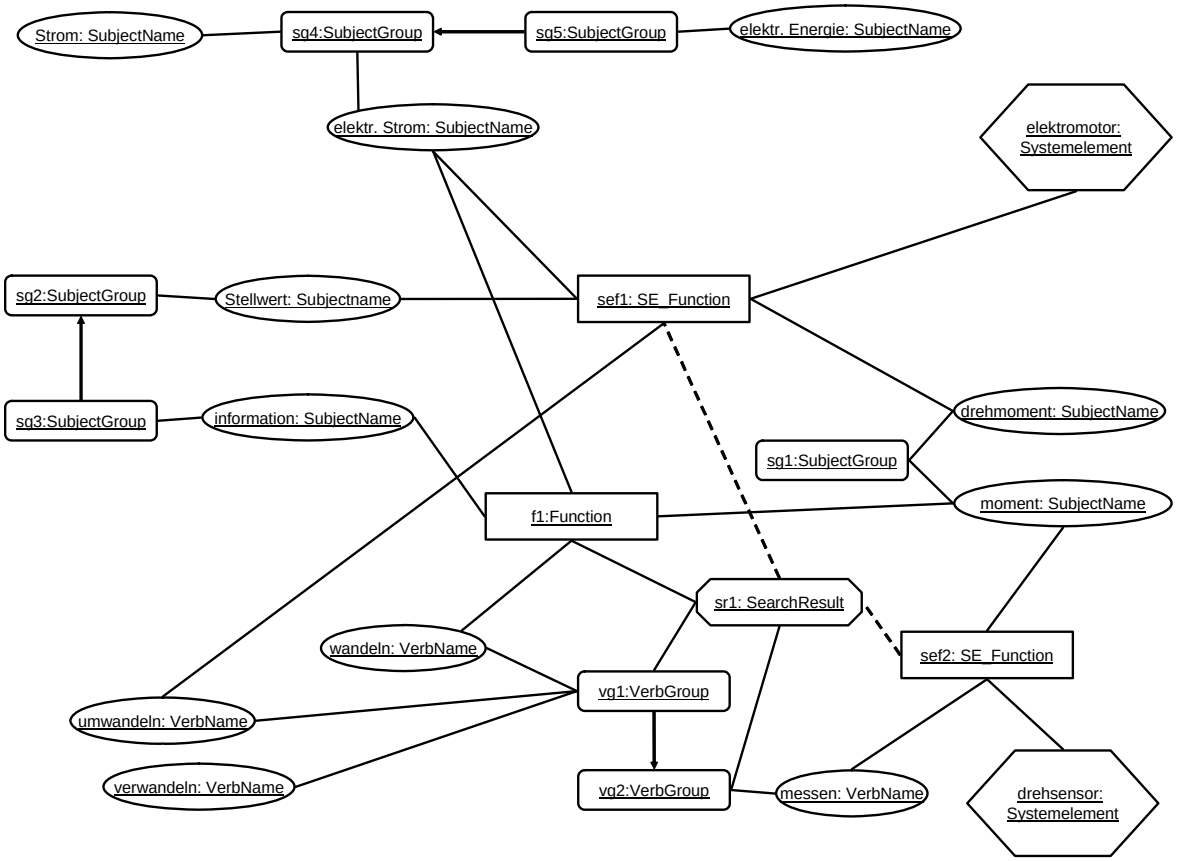


Abbildung 78: Ermitteln möglicher Systemelemente

Berücksichtigung der Input-Substantive

In der Methode checkForInputSubjects(SearchResult) in Story-Pattern vier des Story-Diagramms in Abbildung 68 wird zunächst ermittelt, welche Objekte der Klasse SubjectName über einen functionInput-Link mit dem Objekt f1 verbunden sind. In Abbildung 79 sind dies die Objekte information und elektr. Strom. Für jedes dieser beiden Objekte wird ein neues Objekt der Klasse RequiredSubjects angelegt (rs1 und rs2) und mit dem Objekt sr1 verlinkt. Als nächstes wird festgestellt mit welchem Objekt der Klasse SubjectGroup die Objekte information und elektr. Strom verlinkt sind. Dies sind die Objekte sg3 für das Objekt information und sg4 für das Objekt elektr. Strom. Beide Objekte werden mit dem jeweiligen Objekt der Klasse RequiredSubjects verlinkt (sg3 mit rs1 und sg4 mit rs2). Abschließend werden nun noch alle Objekte der Klasse SubjectGroup ermittelt die über einen Link der Assoziation isParentOf erreicht werden können. Dadurch werden alle Gruppen ermittelt, die die jeweiligen Gruppen konkretisieren. Hier ist es das Objekt sg2, welches mit dem Objekt rs1 verlinkt wird.

In der Methode checkForOutputSubjects(SearchResult) in Story-Pattern vier des Story-Diagramms in Abbildung 68 wird ermittelt, welche Objekte der Klasse SubjectName über einen functionOutput-Link mit dem Objekt f1 verbunden sind. In Abbildung 79 ist dies das Objekt moment. Für dieses Objekte wird ein neues Objekt der Klasse RequiredSubjects angelegt (rs3) und mit dem Objekt sr1 verlinkt. Als nächstes wird festgestellt mit welchem Objekt der Klasse SubjectGroup das Objekte moment verlinkt ist. Dies ist das Objekt sg1. Dieses Objekt wird mit dem Objekt der Klasse RequiredSubjects (rs3) verlinkt.

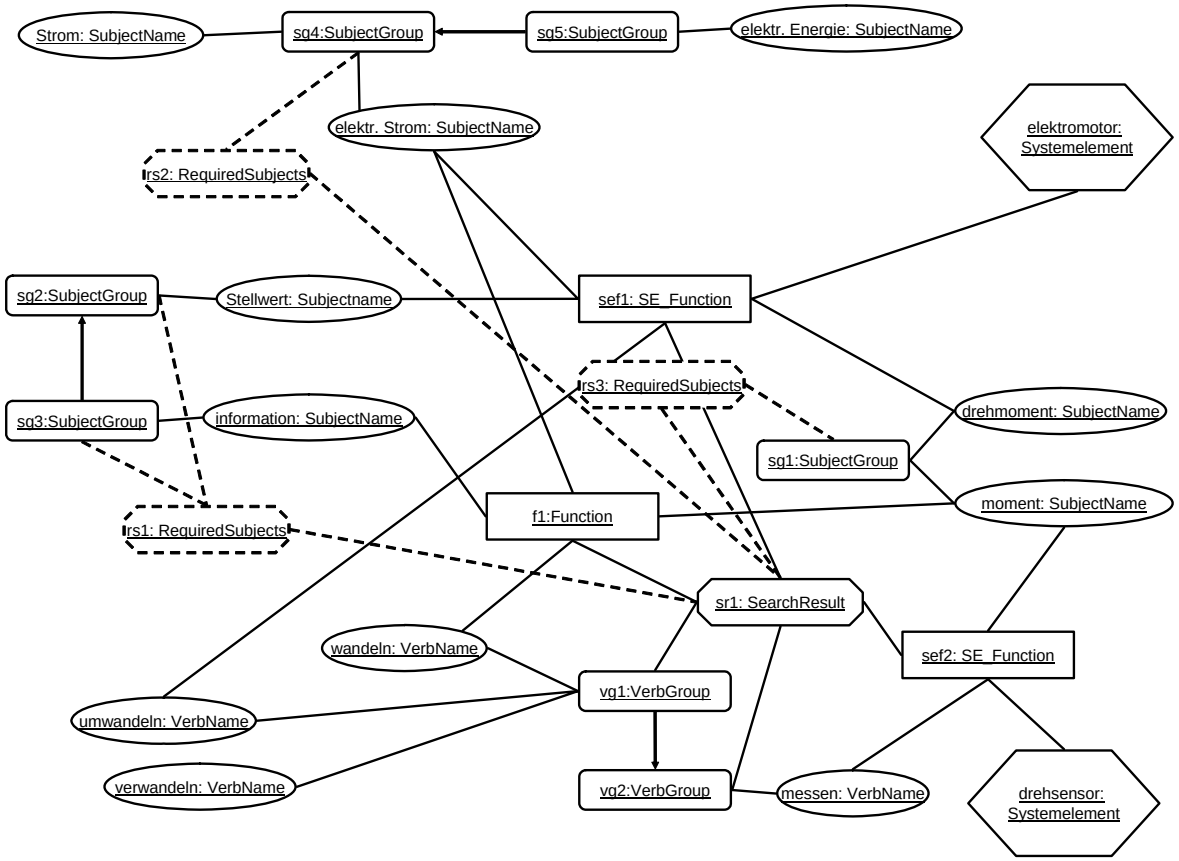


Abbildung 79: Input-Substantive

Entfernen falscher Lösungen

Im letzten Story-Pattern des Story-Diagramms in Abbildung 68 wird innerhalb der Methoden removeWrongElementsInput() und removeWrongElementsOutput() alle falschen Links zwischen dem Objekt sr1 und den Objekten sef1 und sef2 gelöscht. Zu diesem Zweck stellt der Algorithmus fest, ob es einen Pfad gibt vom Objekt sr1 über einen input-Link zu einem Objekt der Klasse RequiredSubjects, von dort zu einem Objekt der Klasse SubjecGroup, weiter zu einem Objekt der Klasse SubjectName, weiter über einen selectedInput-Link zu einem Objekt der Klasse SE_Function und zurück zum Objekt sr1.

Ein solcher Kreis muss für alle Objekte der Klasse RequiredSubjects existieren. Ist dies nicht der Fall, dann wird der Link zwischen dem Objekt sr1 und dem entsprechenden Objekt der Klasse SE_Function gelöscht. In diesem Beispiel konnten für das Objekt sef1 alle Kreise gefunden werden. Für das Objekt sef2 jedoch nicht (vgl. Abbildung 80).

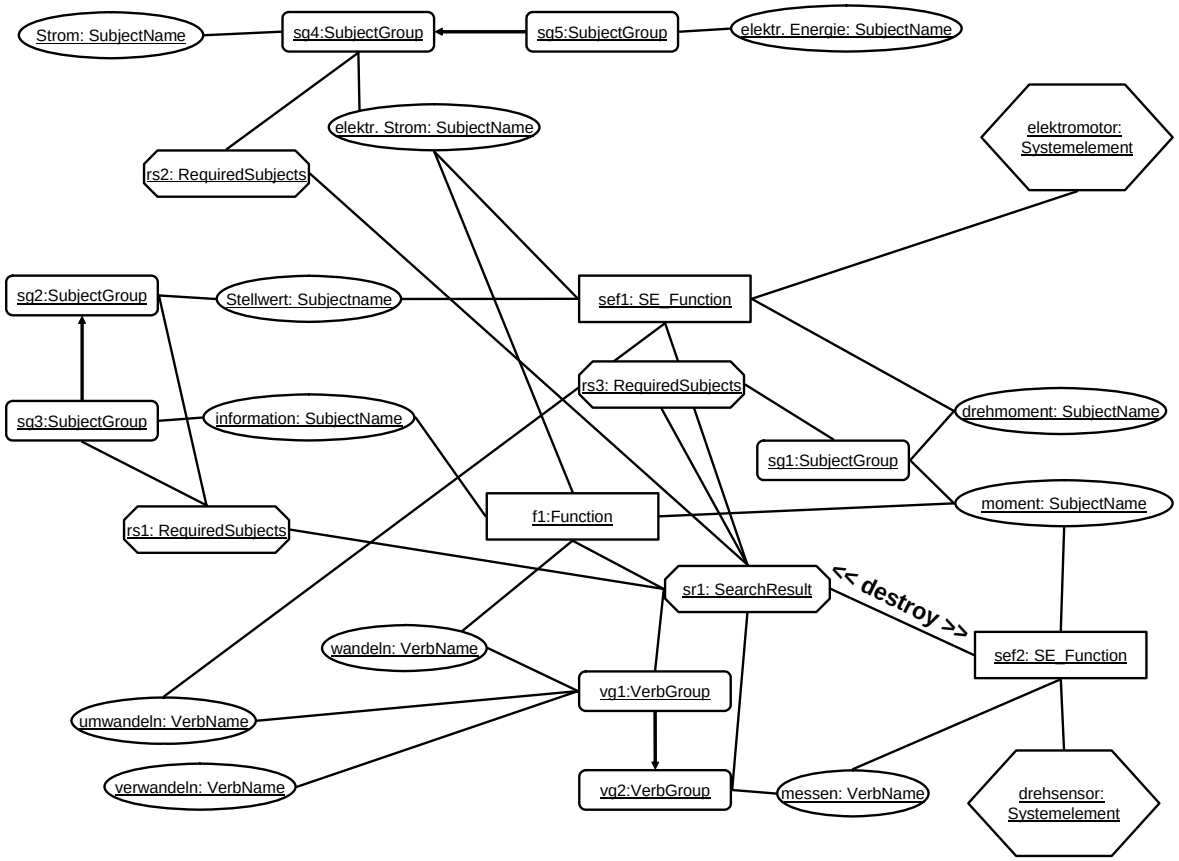


Abbildung 80: Entfernen unnötiger Links

Ergebnis

Nachdem der Algorithmus beendet ist, existiert nur noch ein Link der Assoziation realizedBy (vgl. Abbildung 81). Dieser Link verbindet die Objekte sr1 und sef1 miteinander. Da das Objekt sef1 über einen Link der Assoziation fulfills mit dem Objekt elektromotor verbunden ist, ist dies das einzige Systemelement, das die spezifizierte Funktion f1 inkl. des Verbs und aller Input- und Output-Werte realisiert.

Aus Übersichtlichkeitsgründen wurde in Abbildung 81 bis auf die beiden relevanten Links auf die Anzeige der anderen Links verzichtet.

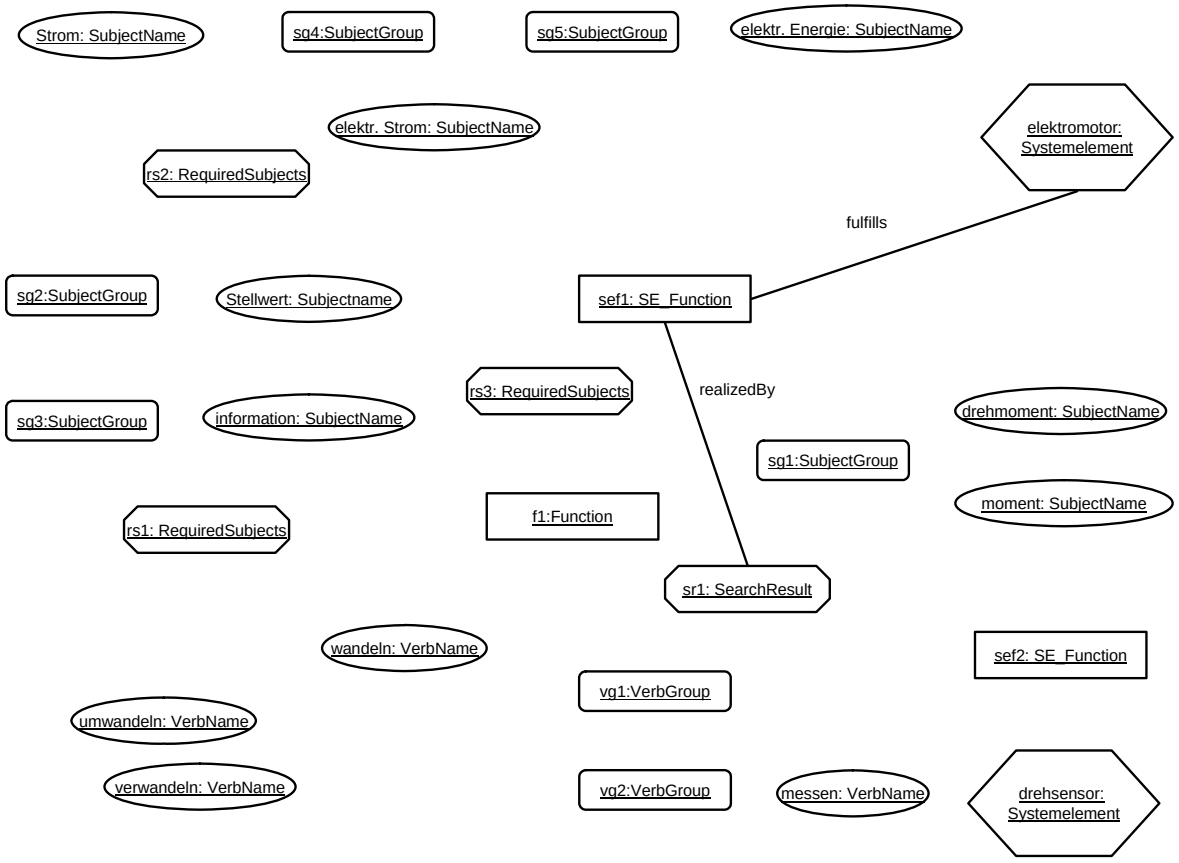


Abbildung 81: Ergebnis

7.6.10 Laufzeitbetrachtung

In diesem Abschnitt wird kurz auf das Laufzeitverhalten des Algorithmus eingegangen, wobei insbesondere der worst-case betrachtet wird. Eine detaillierte Analyse der Laufzeit befindet sich im Anhang dieser Arbeit.

Im worst-case besitzt eine Funktion der Funktionsstruktur (für die ein Systemelement gesucht werden soll) genau ein Verb und alle verfügbaren Substantive sowohl als Input- als auch als Output-Substantive.

Des Weiteren erfüllt jedes Systemelement, das in der Systemelementbibliothek hinterlegt ist alle Funktionen, die sich durch die Kombination der Input- und Output-Substantive ergeben. Die Anzahl möglicher Kombinationen der Substantive ist dabei $2^{|SN|}$ für die Input- und $2^{|SN|}$ für die Output-Substantive[30]. Für ein einzelnes Verb ergeben sich somit $2^{2*|SN|}$ Funktionen die von einem Systemelement erfüllt werden.

Gibt es beispielsweise 5 Verben und 10 Substantive, dann gibt es pro Verb 1.048.576 verschiedene Möglichkeiten die Input- und Output-Substantive zu kombinieren. Insgesamt gibt es dann 5.242.880 verschiedene Funktionen die ein Systemelement erfüllt. Im worst-case bedeutet dies, dass all diese Funktionen mit der gesuchten Funktion verglichen werden müssen.

Die Analyse zeigt, dass die Mengen der Verben und der Substantive entscheidenden Einfluss auf die Komplexität und damit auf die Laufzeit haben. Insbesondere zeigt die Analyse, dass die Laufzeit des Algorithmus exponentiell von der Anzahl der Substantive abhängt.

[30] |SN| = Anzahl der Substantive

Konzeptionelle Betrachtung der Laufzeit

Die Laufzeitbetrachtung beruht insbesondere auf der Annahme, dass die gesuchte Funktion alle verfügbaren Substantive als Input und alle verfügbaren Substantive als Output besitzt. Ferner geht die Abschätzung davon aus, dass sowohl jedes Verb mit allen anderen Verben untereinander in Beziehung steht, als auch jedes Substantiv mit allen anderen Substantiven in Beziehung steht. Des Weiteren wird angenommen, dass jedes Verb eine eigene Verbgruppe und jedes Substantiv eine eigene Substantivgruppe besitzt. Schließlich wird angenommen, dass jedes Systemelement jede mögliche Kombination von Funktionen erfüllt.

Diese Annahmen stellen das absolute worts-case Szenario dar. Unter der Voraussetzung, dass es eine ausreichend große Menge an Substantive und Verben gibt, kommt dieses Szenario im praktischen Einsatz nicht vor.

Folgende Erkenntnisse sind daher entscheidend für die Beurteilung des Algorithmus:

(1) Es gibt keine vollständig zusammenhängenden Mengen von Verben und Substantiven. Es gibt immer nur kleinere Gruppen.

(2) Die Funktionen einer Funktionshierarchie haben eine wesentlich geringere Anzahl an Input- und Output-Substantive, als im worst-case Szenario angenommen.

(3) Die Systemelemente erfüllen in der Regel eine deutlich geringere Anzahl an unterschiedlichen Funktionen.

(4) Nicht jedes Verb bzw. Substantiv stellt eine eigene Gruppe dar. Dies würde bedeuten, dass es keine äquivalenten Verben bzw. Substantive gibt, was die Untersuchungen innerhalb dieser Arbeit jedoch ausschließen.

Aufgrund dieser Erkenntnisse kann man im durchschnittlichen Fall von einer Laufzeit ausgehen, die für die meisten Entwickler akzeptabel sein sollte.

7.7 Zusammenfassung

Im vorliegenden Kapitel wurde ein Algorithmus vorgestellt, mit dessen Hilfe Systemelemente automatisch ermittelt werden. Ausgehend von einer Funktionshierarchie wird für jede Funktion explizit nach Systemelementen gesucht, die diese Funktion erfüllen können.

Zuerst wurde die Ausgangssituation vorgestellt, gefolgt von den Anforderungen an den Algorithmus. Dann wurde auf abstrakter Ebene die Funktionsweise des Algorithmus präsentiert, gefolgt von einer formalen Beschreibung mit Hilfe von Graphtransformationsregeln. Schließlich wurde eine Abschätzung der Laufzeit vorgenommen mit dem Ergebnis, dass der Algorithmus im Mittel eine akzeptable Laufzeit annimmt.

KAPITEL 8: KONSISTENZ DER MODELLE

In den letzten Kapiteln wurden die Funktionshierarchie, die Systemstruktur und die automatische Suche nach Systemelementen erläutert. In diesem Kapitel wird nun gezeigt, wie Stück für Stück die Systemstruktur entsteht, wobei insbesondere der konzeptionelle Zusammenhang zwischen der Funktionshierarchie und der Systemstruktur im Vordergrund steht.

Systemelemente einer Systemstruktur werden verwendet, um Funktionen einer Funktionshierarchie zu realisieren. Hierzu werden die Systemelemente der jeweiligen Funktion zugewiesen.

Während des Entwurfs, also während der Modellierung der Funktionshierarchie im Wechselspiel mit der Systemstruktur, kann es häufig zu Änderungen der beiden Modelle kommen. Diese Änderungen können dazu führen, dass der Überblick darüber verloren geht, welche Funktionen bereits von Systemelementen realisiert werden und welche noch nicht. Um dies zu verhindern, müssen die Änderungen entsprechend überwacht werden.

Zu diesem Zweck werden in diesem Kapitel entsprechende Konsistenzbedingungen definiert und mit Hilfe von Graphtransformationsregeln formalisiert.

8.1 Abhängigkeiten zwischen Funktionshierarchie und Systemstruktur

In diesem Abschnitt werden die Konsistenzbedingungen vorgestellt, die zwischen der Funktionshierarchie und der Systemstruktur definiert wurden. Ferner werden die Operationen vorgestellt, die diese Konsistenzbedingungen sicherstellen. Schließlich wird Anhand der Stati der Funktionen gezeigt, wie der Realisierungsgrad der Funktionen bestimmt wird und welches Datenmodell dem Ansatz zugrunde liegt.

8.1.1 Konsistenzbedingungen zwischen der Funktionshierarchie und der Systemstruktur

Der Übergang von der Funktionshierarchie zur Systemstruktur ist ein entscheidender Schritt während des Systementwurfs [Kal98], [VDI04]. Wichtig hierbei ist, dass jede Funktion der Funktionshierarchie von einem oder mehreren Systemelementen realisiert wird. Erst wenn alle Funktionen durch Systemelemente realisiert werden, ist der Systementwurf fertig und die prinzipielle Lösung steht fest.

Aufgrund des dynamischen Umfelds und sich ändernder Anforderungen unterliegen sowohl die Funktionshierarchie, als auch die Systemstruktur ständigen Änderungen. So kann z.B. das Löschen eines Systemelements bedeuten, dass eine Funktion nicht mehr realisiert wird, oder das aufgrund des Löschens einer Funktion verschiedene Systemelemente obsolet werden. Diese Änderungen müssen erkannt und dem Entwickler mit Hilfe geeigneter Mechanismen transparent gemacht werden.

Die Sicherstellung der Konsistenz zwischen Funktionshierarchie und Systemstruktur mit Hilfe eines Konsistenzmechanismus erfordert es, dass entsprechende Konsistenzbedingungen definiert werden. Diese lauten:

(1) Jedes Systemelement muss mindestens einer Funktion zugewiesen sein, wenn eine entsprechende Funktionshierarchie existiert.

(2) Wenn ein Systemelement Hilfsfunktionen besitzt, dann müssen diese in der Funktionshierarchie vorhanden sein.

(3) Hilfsfunktionen eines Systemelements dürfen nicht in der Funktionshierarchie enthalten sein, wenn das Systemelement nicht in der Systemstruktur enthalten ist.

(4) Es muss jederzeit nachvollzogen werden können, ob eine Funktion bereits durch ein Systemelement realisiert wird, oder nicht.

8.1.2 Konsistenzmechanismus

Im letzten Abschnitt wurde gezeigt, welche Konsistenzbedingungen existieren, die mit Hilfe eines geeigneten Mechanismus eingehalten werden müssen. In der Literatur sind verschiedene Ansätze zur Sicherstellung der Konsistenz zwischen Modellen bekannt.

Küster [Küs04], Grosse-Rhode [Gros01] oder Zave und Jackson [ZJ93] definieren Konsistenz zwischen Modellen, indem sie verschiedene Modelle auf eine gemeinsame semantische Domäne abbilden. Moreira and Clark [MC94] beschreiben Konsistenz zwischen Modellen, indem sie insbesondere Modelle der UML mit Hilfe der Sprache LOTOS formalisieren. Weitere Ansätze beschreiben Engels und Schäfer [ES89], Nagl [Nagl96] oder Perin [Peri00], indem sie zur Sicherstellung der Konsistenz, Graphstransformationsregeln verwenden. Diese Idee, die Sicherung der Konsistenz auf Basis von Graphtransformationsregeln, wird auch im Rahmen dieser Arbeit verwendet.

Die statische Struktur sowohl der Funktionshierarchie, als auch der Systemstruktur wird mit Hilfe eines UML-Klassendiagramms formalisiert. Innerhalb des Klassendiagramms sind verschiedene Klassen durch Assoziationen miteinander verknüpft. Diese Assoziationen drücken z.B. aus, dass ein Systemelement einer Funktion zugewiesen werden kann.

Während der konkreten Modellierung der Funktionshierarchie und der Systemstruktur werden Klassen instanziiert, wodurch entsprechende Objekte entstehen. Gleichzeitig werden auch Assoziationen instanziiert (Links), wodurch dann ein Graph von verlinkten Objekten, der Objektgraph, entsteht.

Das Erstellen und Ändern des Objektgraphen, wie beispielsweise das Löschen von Objekten (z.B. eines Systemelements), oder das Ändern von Links, erfolgt mit Hilfe unterschiedlicher Operationen. Diese Operationen werden in Form von Methoden der jeweiligen Klassen beschrieben. Durch diese Methoden wird sicher gestellt, dass alle zur Modellierung notwendigen Operationen (vgl. 8.1.3) möglich sind und darüber hinaus die oben beschriebenen Konsistenzbedingungen eingehalten werden.

Formal wird das Verhalten der Methoden, mit Hilfe von Graphtransformationsregeln in Form von Story-Diagrammen beschrieben (Näheres zu Story-Diagrammen, siehe Kapitel 7.5).

8.1.3 Operationen zur Sicherstellung der Konsistenz zwischen Funktionshierarchie und Systemstruktur

Wie im letzten Abschnitt erwähnt, wird die Konsistenz mit Hilfe unterschiedlicher Operationen sichergestellt. Diese Operationen müssen die Modellierung der Funktionshierarchie bzw. der Systemstruktur, gemäß den in Kapitel fünf und sechs beschriebenen Angaben, ermöglichen. Ferner müssen sie die oben beschriebenen Konsistenzbedingungen sicherstellen.

Folgende Operationen sind für die Bearbeitung der Funktionshierarchie und der Systemstruktur notwendig bzw. zulässig:

Funktionshierarchie:

(1) Anlegen von Funktionen (vgl. 8.2)

(2) Löschen von Funktionen (vgl. 8.3)

(3) Verschieben von Funktionen (vgl. 8.4)

(4) Setzen eines Status (vgl. 8.13)

Systemstruktur:

(1) Anlegen von Systemelementen (vgl. 8.5)

(2) Löschen von Systemelementen (vgl. 8.11)

(3) Zuweisen von Systemelementen zu Funktionen (vgl. 8.6)

(4) Ergänzen von Hilfsfunktionen (vgl. 8.7)

(5) Löschen von Zuweisungen (vgl. 8.8)

(6) Verschieben von Zuweisungen (vgl. 8.9)

(7) Verschieben von Hilfsfunktionen (vgl. 8.10)

(8) Löschen von Hilfsfunktionen (vgl. 8.12)

8.1.4 Status einer Funktion

Jeder Funktion innerhalb der Funktionshierarchie werden im Laufe des Entwurfs ein oder mehrere Systemelemente vom Entwickler zugewiesen, um sie zu realisieren. Bei einigen Funktionen kann es vorkommen, dass sie erst durch die Zuweisung mehrerer Systemelemente realisiert werden. Von entscheidendem Interesse ist also, ob eine Funktion bereits vollständig, teilweise oder noch gar nicht realisiert wird.

Um dies für jede Funktion festlegen zu können, wurden drei verschiedene Stati definiert:

- nicht realisiert
- teilweise realisiert
- realisiert

Die Vergabe dieser Stati kann entweder automatisch, oder manuell durch den Entwickler erfolgen.

Automatische Statusvergabe

Die automatische Statusvergabe wird immer dann durchgeführt, wenn eine der oben beschriebenen Operationen durchgeführt wurde. Nach jeder Modifikation der Funktionshierarchie, oder der Systemstruktur werden die Stati überprüft und gegebenenfalls angepasst. Die Grundlage für die automatische Statusvergabe liefern dabei entsprechende Regeln (vgl. „Regeln zur Statusvergabe“).

Für weitere Details sei auf Abschnitt 8.13.2 verwiesen.

Manuelle Statusvergabe

Neben der automatischen Statusvergabe ist auch eine manuelle Vergabe der Stati möglich. Dadurch haben die Entwickler jeder Zeit die Kontrolle darüber, welchen Status eine Funktion hat. Ein einmal manuell vergeben Status kann durch die automatische Statusvergabe nicht mehr geändert werden.

Für weitere Details sei auf Abschnitt 8.13.1 verwiesen.

Regeln zur Statusvergabe

Um eine automatische Statusvergabe zu ermöglichen, müssen Regeln definiert werden die angeben, in welcher Situation welcher Status zu vergeben ist.

Zu diesem Zweck wurden die Funktionen in zwei Gruppen unterteilt. In der ersten Gruppe befinden sich alle Funktionen, die nicht weiter in Teilfunktionen zerlegt wurden (Blätter der Funktionshierarchie). In der zweiten Gruppe befinden sich alle anderen Funktionen.

Für Funktionen ohne Teilfunktion gelten folgende Regeln für die Statusvergabe:

Nr.	Status	Bedeutung
1	realisiert	Der Funktion ist mindestens ein Systemelement zugewiesen und dieses Systemelement wurde automatisch aus der Bibliothek ermittelt
2	teilweise realisiert	Der Funktion ist mindestens ein Systemelement zugewiesen.
3	nicht realisiert	Der Funktion ist kein Systemelement zugewiesen.

Tabelle 1: Statusvergabe für Funktionen ohne Teilfunktionen

Hinweis zum Status „realisiert“: Aufgrund der Tatsache, dass man einer Funktion mehrere Systemelemente zuweisen kann und diese Systemelemente nicht aus der Systemelementbibliothek stammen müssen, ist eine automatische Statusvergabe nicht immer möglich. Sobald einer Funktion, die selbst keine eigenen Teilfunktionen besitzt, ein Systemelement zugewiesen wurde, das nicht vorher durch den in Kapitel 7 vorgestellten Algorithmus ermittelt wurde, gibt es keine Möglichkeit festzustellen, ob diese Zuweisung ausreicht, um die Funktion vollständig zu realisieren. In diesem Fall ist die manuelle Vergabe das Status durch den Entwickler erforderlich.

Für Funktionen mit Teilfunktion gelten folgende Regeln für die Statusvergabe:

Nr.	Status	Bedeutung
1	realisiert	Alle Teilfunktionen haben den Status „realisiert“.
2	teilweise realisiert	1. Mindestens ein Systemelement wurde der Funktion zugewiesen oder 2. Mindestens eine Teilfunktion hat den Status „teilweise realisiert“ oder 3. Mindestens eine Teilfunktion hat den Status „realisiert“ und mindestens eine Teilfunktion hat den Status „nicht realisiert“.
3	nicht realisiert	Alle Teilfunktionen haben den Status „nicht realisiert“.

Tabelle 2: Statusvergabe für Funktionen mit Teilfunktionen

Abbildung 82 zeigt beispielhaft eine Funktionshierarchie mit entsprechenden Stati. Teilfunktion (1) wird nur teilweise realisiert, da die Teilfunktion (3) noch gar nicht realisiert wird. Teilfunktion (4) wird noch nicht realisiert, da noch kein Systemelement zugewiesen wurde. Teilfunktion (5) wird vollständig realisiert, da die Teilfunktionen (6) und (7) vollständig realisiert werden. Die Gesamtfunktion wird nur teilweise realisiert, da noch nicht alle Teilfunktionen vollständig realisiert werden.

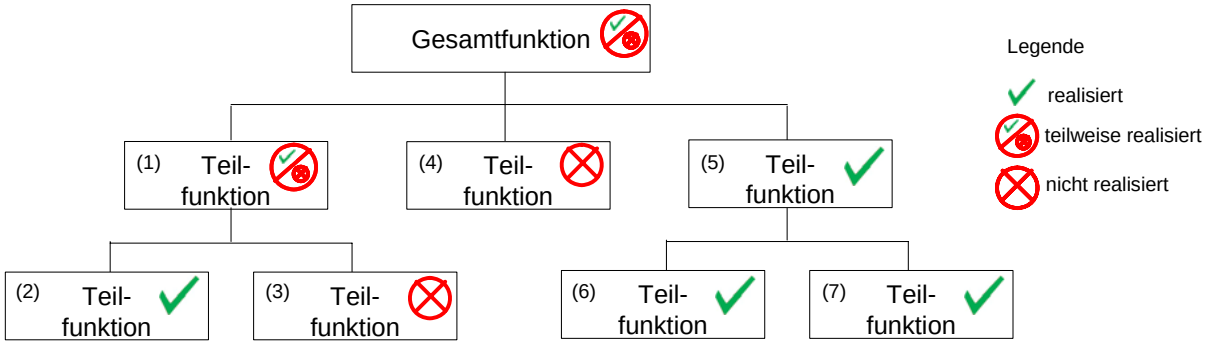


Abbildung 82: Statusvergabe innerhalb der Funktionshierarchie

8.1.5 Datenstruktur

Bevor auf die unter 8.1.3 erwähnten Operationen näher eingegangen werden kann, ist es erforderlich ein entsprechendes Datenmodell zu definieren. Nur mit Hilfe eines solchen Datenmodells ist die Verhaltensbeschreibung der Operationen mit Hilfe von Story-Diagrammen (Graphtransformationsregeln) möglich.

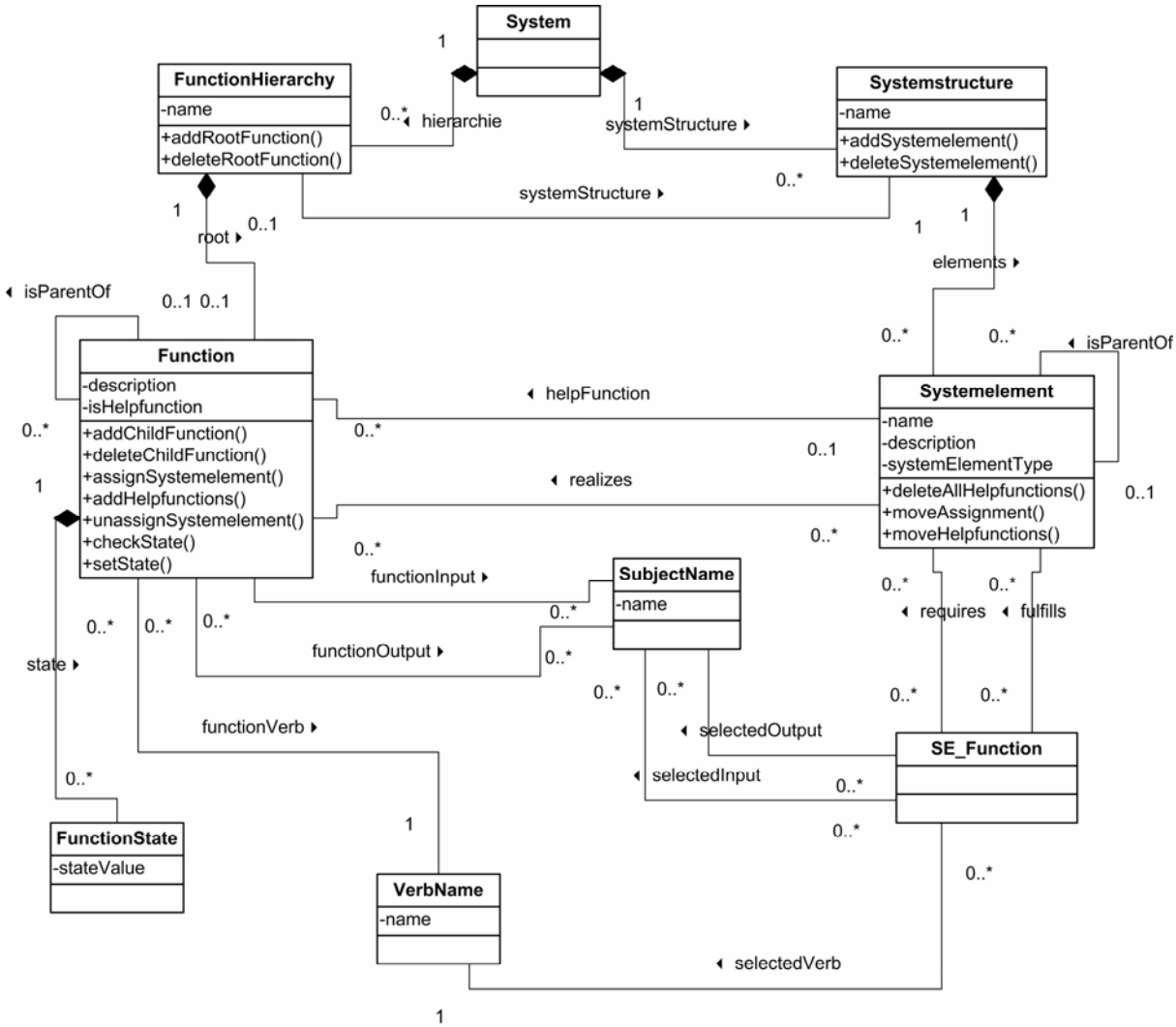


Abbildung 83: UML-Klassendiagramm für die Konsistenzkontrolle

Kurzbeschreibung der Klassen:

System: Siehe Kapitel 6.3.1

FunctionHierarchy: Siehe Kapitel 5.7. Zusätzlich gibt es die Assoziation systemStructure. Dadurch wird sicher gestellt, dass die zur Funktionshierarchie gehörende Systemstruktur erreicht wird. Zu einer Funktionshierarchie muss es immer auch eine Systemstruktur geben. Zu einer Systemstruktur kann eine Funktionshierarchie existieren.

Systemstructure: Siehe Kapitel 6.3.1

Function: Siehe Kapitel 5.7. Neu sind hier die Assoziationen realizes und helpFunctions. Mit der Assoziation realizes wird die „Erfüllungsbeziehung“ zwischen Systemelementen und Funktionen spezifiziert (näheres zu Zuweisungen, siehe 8.6). Die Assoziation helpFunctions zeigt an, dass die assoziierte Funktion eine Hilfsfunktion des Systemelements ist.

FunctionState: Diese Klasse beschreibt den Status der Funktion. Mit Hilfe des Attributs stateValue werden die Stati festgelegt. Der Wert -1 bedeutet „nicht realisiert“, 0 bedeutet „teilweise realisiert“ und 1 bedeutet „realisiert“.

SystemElement: Siehe Kapitel 6.3.1

SE_Function: Siehe Kapitel 6.3.1

VerbName: Siehe Kapitel 6.3.1

SubjectName: Siehe Kapitel 6.3.1

Auf Basis dieses Datenstruktur wird im weiteren mit Hilfe von Story-Diagrammen bzw. Story-Pattern das Verhalten der oben angegebenen Operationen formal beschrieben.

8.2 Anlegen von Funktionen

In den folgenden Abschnitten wird gezeigt, wie eine Funktionshierarchie erstellt wird. Zuerst wird die Erstellung der Gesamtfunktion und im Anschluss die Erstellung von Teilfunktionen erläutert.

8.2.1 Anlegen einer Gesamtfunktion

Zu Beginn der Modellierung wird typischerweise eine Funktionshierarchie erstellt, wobei die erste Funktion die Gesamtfunktion ist. Sie stellt die Wurzel der Funktionshierarchie dar.

Die durchzuführenden Aktionen sind:

(1) Anlegen der Gesamtfunktion

(2) Zuweisen bereits vorhandener Systemelemente zur Gesamtfunktion (falls erforderlich)

Formale Beschreibung zur Erstellung der Gesamtfunktion

Das Anlegen einer neuen Gesamtfunktion wird mit Hilfe der Methode addRootFunction der Klasse FunctionHierarchy durchgeführt (vgl. Abbildung 84). In Schritt (1) wird die Gesamtfunktion (root:Function - Objekt) erstellt und mit der Funktionshierarchie (this) verknüpft. Zusätzlich wird ein Statusobjekt (functionState-Objekt) erzeugt und mit der gerade erzeugten Gesamtfunktion (root-Objekt) verlinkt. Der Status der Funktion wird initial mit dem Wert „nicht realisiert“ belegt. Dies drückt sich im Attribut stateValue des functionState-Objektes aus.

Die folgenden Schritte (2) bis (4) sind nur notwendig, falls bereits eine Systemstruktur mit verschiedenen Systemelementen existiert. In diesem Fall müssen die bereits vorhandenen Systemelemente der Gesamtfunktion zugewiesen werden, damit mögliche Hilfsfunktionen berücksichtigt werden und die Konsistenzbedingung 1 aus Abschnitt 8.1.1 sicher gestellt wird.

Die Möglichkeit eine Systemstruktur zu modellieren, ohne gleichzeitig eine Funktionshierarchie zu modellieren, erlaubt dem Entwickler eine gewisse Flexibilität. In einigen Fällen kann die Systemstruktur so klein sein (wenige Systemelemente), dass eine Funktionshierarchie nicht nötig ist. Von Seiten des Autors wird jedoch empfohlen zu jeder Systemstruktur auch eine Funktionshierarchie anzulegen.

In Schritt (2) der Abbildung 84 wird die zur Funktionshierarchie zugehörige Systemstruktur ermittelt (systemStructure:SystemStructure). Dies ist notwendig, damit bereits vorhandene Systemelemente der neuen Gesamtfunktion zugewiesen werden können. Die Ermittlung bereits vorhandener Systemelemente erfolgt in Schritt (3). Der doppelte Rahmen gibt dabei an, dass alle vorhandenen Systemelemente betrachtet werden und nicht nur eins. Jedes der in diesem Schritt gebundenen Systemelemente (systemelement-Objekte) wird der Gesamtfunktion zugewiesen. Diese Zuweisung erfolgt in Schritt (4) mit Hilfe der Methode assignSystemelement.

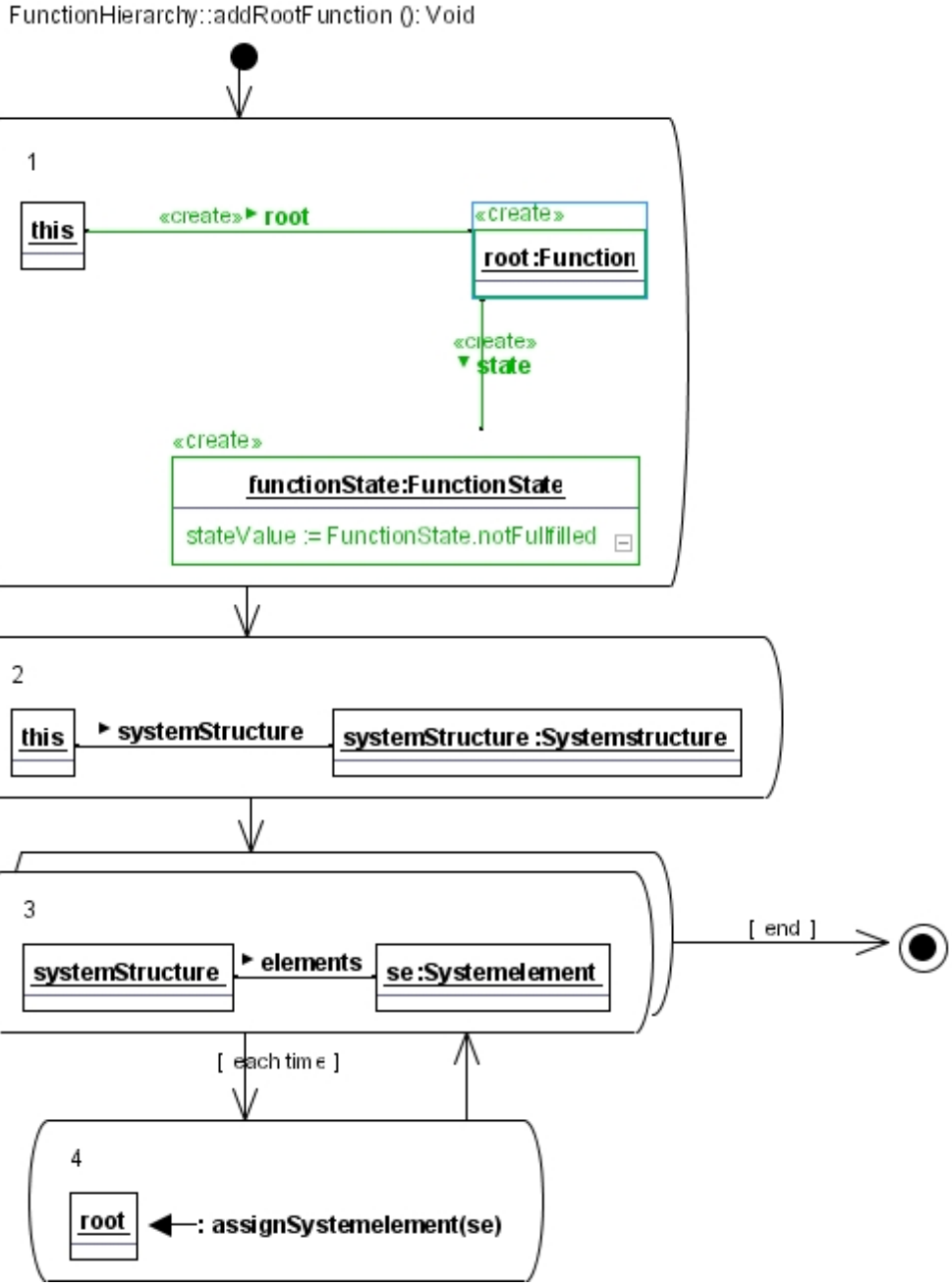


Abbildung 84: Erstellen der Gesamtfunktion

Beispiel

Das folgende Beispiel (Abbildung 85) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

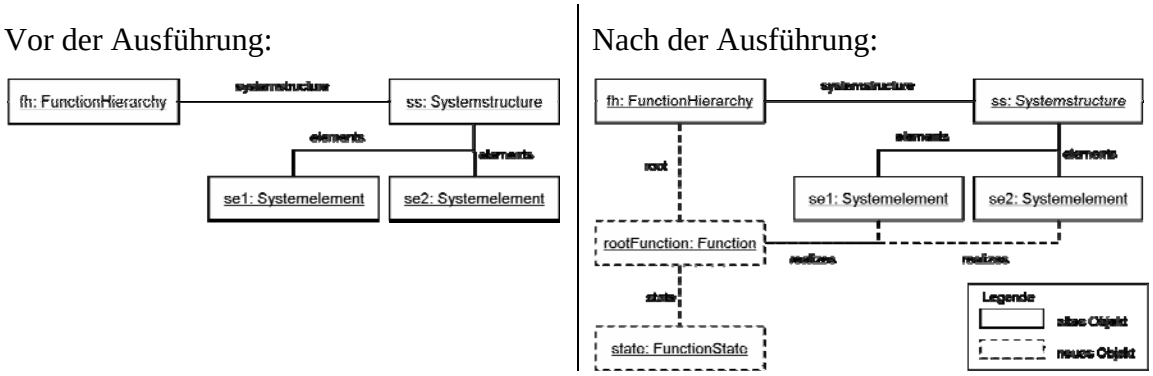


Abbildung 85: Anwenden der Graphtransformationsregel "addRootFunction"

8.2.2 Anlegen und Anordnen von Teilfunktionen

Nachdem eine Gesamtfunktion angelegt wurde, können nun Teilfunktionen erzeugt und in der Funktionshierarchie angeordnet werden. Eine Teilfunktion stellt dabei die Verfeinerung einer anderen Funktion dar, entweder der Gesamtfunktion oder einer anderen Teilfunktion (vgl. Kapitel 5.3).

Prinzipiell gibt es zwei Aktionen, die durchgeführt werden können. Dies sind:

(1) Anlegen einer Funktion

(2) Anordnen einer Funktion als Teilfunktion einer anderen Funktion

Formale Beschreibung zur Erstellung einer Teilfunktion

Die Ausführung der Aktionen erfolgt mit Hilfe der Methode addChildFunction der Klasse Function (vgl. Abbildung 86). Wird der Methode ein Objekt der Klasse Function übergeben, dann soll diese übergebene Funktion als Teilfunktion angeordnet werden. Dies kommt beim Verschieben von Funktionen innerhalb der Funktionshierarchie zur Anwendung. Wird kein Objekt übergeben, dann soll eine neue Funktion erzeugt werden.

Zu Beginn der Methode wird überprüft, ob als Übergabeparameter eine Funktion (Function-Objekt) übergeben wurde. Ist dies der Fall (function != null), dann wird mit Schritt (1) fortgefahren, in dem die übergebene Funktion (function) mit der aktuellen Funktion (this) verlinkt wird (<<create>> isParentOf). Dies Verlinkung besagt, dass die übergebene Funktion jetzt eine Teilfunktion der Funktion ist, auf der die Methode aufgerufen wurde.

Schritt(2) wird ausgeführt, wenn kein function-Objekt der Methode übergeben wurde. Jetzt wird eine neue Funktion (newFunction:Function) erzeugt und mit der aktuellen Funktion (this) verlinkt. Des Weiteren wird ein Statusobjekt (functionState:FunctionState) erzeugt. Dieses wird mit dem Status „nicht realisiert“ initialisiert (stateValue:=FunctionState.notFullfilled) und mit der neuen Funktion (newFunction) verlinkt (<<create>> state).

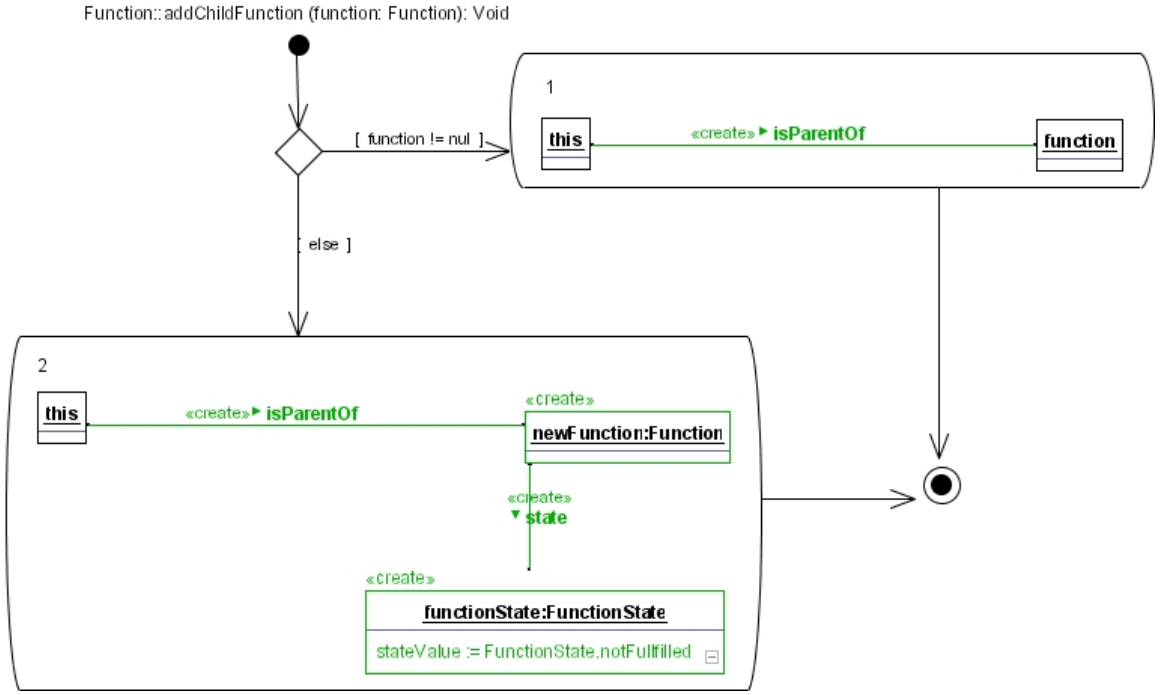


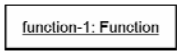
Abbildung 86: Anlegen einer Teilfunktion

Beispiel

Das folgende Beispiel (Abbildung 87) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel wurde der Methode keine Funktion als Parameter übergeben, d.h. es wurde StoryPattern (2) ausgeführt und eine neue Funktion (function-2) inkl. einem entsprechendem Statusobjekt (state) angelegt und mit der ursprünglichen Funktion (function-1) verlinkt.

Vor der Ausführung:



Nach der Ausführung:

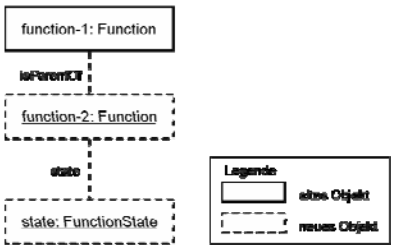


Abbildung 87: Anwenden der Graphtransformationsregel "addChildFunction"

8.3 Löschen von Funktionen

Nachdem in Abschnitt 8.2 gezeigt wurde, wie Funktionen angelegt werden, wird in diesem Abschnitt erläutert, wie Funktionen wieder gelöscht werden können. Zuerst wird das löschen einer Teilfunktion und dann das Löschen der Gesamtfunktion beschrieben.

8.3.1 Löschen einer Teilfunktion

Löschen von Teilfunktionen ist in verschiedenen Situationen erforderlich. Insbesondere aufgrund von Anforderungsänderungen oder aufgrund neuer Ideen bei der Strukturierung der Funktionshierarchie kann das Löschen einer Teilfunktion notwendig werden.

Das Löschen von Teilfunktionen kann zum einen manuell durch den Entwickler und zum anderen automatisch erfolgen. Insbesondere das automatische Löschen ist erforderlich, da nur so die Konsistenzbedingungen aus Abschnitt 8.1.1 sicher gestellt werden können. Wird z.B. ein Systemelement gelöscht, dann müssen auch seine Hilfsfunktionen gelöscht werden, die u. U. in der Funktionshierarchie vorhanden sind (vgl. Konsistenzbedingung 3). Des Weiteren muss sicher gestellt werden, dass diejenigen Systemelemente, die diesen Hilfsfunktionen evtl. bereits zugewiesen wurden, auch weiterhin einem Systemelement zugewiesen sind (vgl. Konsistenzbedingung 1).

Insgesamt ergeben sich folgende Aktionen:

(1) Ermitteln und Löschen aller Teilfunktionen der zu löschenden Funktion

(2) Verschieben bereits vorhandener Zuweisungen zur Gesamtfunktion

(3) Löschen der Funktion

Formale Beschreibung zum Löschen einer Teilfunktion

Das Löschen einer Teilfunktion erfolgt, indem auf der übergeordneten Funktion die Methode deleteChildFunction aufgerufen wird (vgl. Abbildung 88). Als Übergabeparameter wird die zu löschende Funktion (deleteFunction) übergeben.

In Schritt (1) wird zunächst überprüft, ob die zu löschende Funktion selbst noch Teilfunktionen besitzt (isParentOf-Link). Für jede so ermittelte Teilfunktion (deleteFunction2:Function) wird dann in Schritt (2) wieder die Methode deleteChildFunction aufgerufen. Die so entstehende Rekursion sorgt dafür, dass alle weiteren Teilfunktionen, d.h. der gesamte Teilbaum unterhalb der zu löschenden Funktion, ebenfalls gelöscht werden.

Wurden alle Teilfunktionen gelöscht, oder keine gefunden, dann werden in Schritt (3) alle Systemelemente (se:Systemelement) ermittelt, die der zu löschenden Funktion (deleteFunction) zugewiesen wurden. Konnten entsprechend zugewiesene Systemelemente ermittelt werden, dann müssen diese Zuweisungen aufgehoben werden und der Gesamtfunktion zugewiesen werden. Dies erfolgt in Schritt (4), indem die Methode unassignSystemelement(se) auf der zu löschenden Teilfunktion aufgerufen wird (vgl. 8.8). Auf diese Weise wird sicher gestellt, dass die Konsistenzbedingung 1 eingehalten wird.

Nachdem alle Teilfunktionen gelöscht und alle zugewiesenen Systemelemente entfernt wurden, wird in Schritt (5) die eigentliche Funktion gelöscht. Hierbei wird nicht nur das deleteFunction-Objekt, sondern auch das zugehörige Status-Objekt (functionState-Objekt) gelöscht.

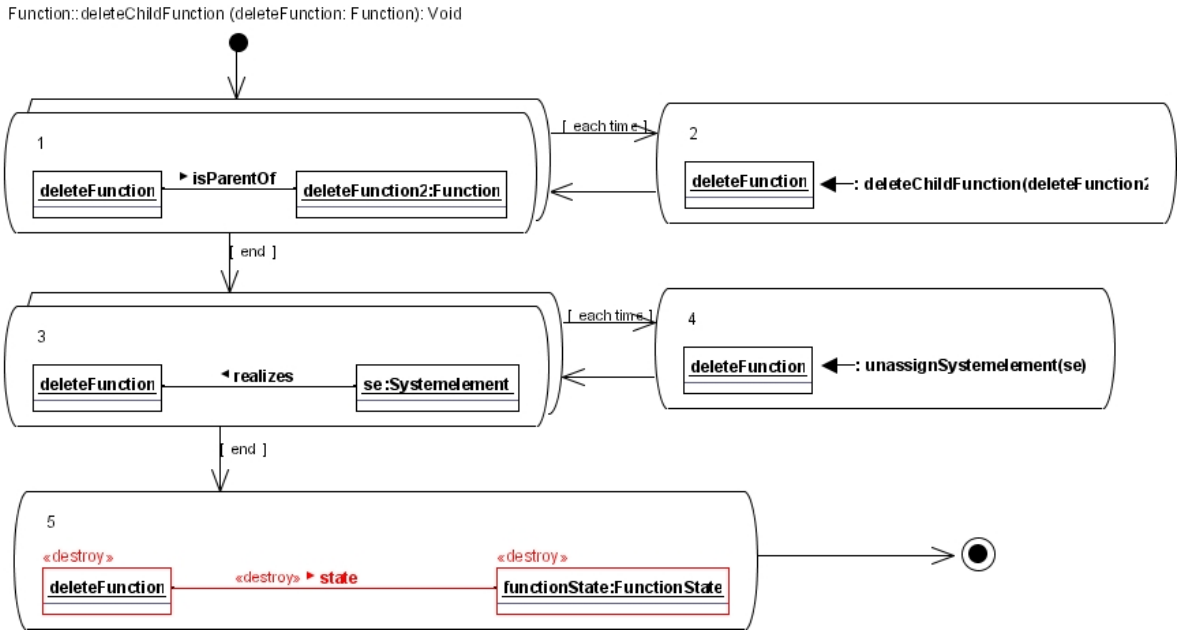


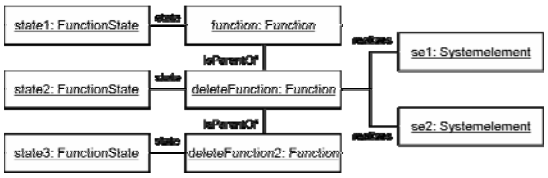
Abbildung 88: Löschen einer Teilfunktion

Beispiel

Das folgende Beispiel (Abbildung 89) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel werden zuerst alle Teilfunktionen der zu löschenden Funktion (deleteFunction2) inkl. des zugehörigen Statusobjektes (state3) gelöscht. Im Anschluss werden die Verknüpfungen zu den Systemelementen (se1 und se2) und schließlich die Funktion (deleteFunction) inkl. ihres Statusobjektes (state2) gelöscht.

Vor der Ausführung:



Nach der Ausführung:

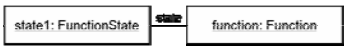


Abbildung 89: Anwenden der Graphtransformationsregel "deleteChildFunction"

8.3.2 Löschen der Gesamtfunktion

Nachdem im Abschnitt 8.3.1 gezeigt wurde, wie einzelne Teilfunktionen gelöscht werden, wird in diesem Abschnitt das Löschen der Gesamtfunktion beschrieben.

Die durchzuführenden Aktionen sind:

(1) Ermitteln und löschen aller Teilfunktionen

(2) Löschen der Gesamtfunktion

Formale Beschreibung zum Löschen der Gesamtfunktion

Das Löschen der Gesamtfunktion erfolgt mit Hilfe der Methode deleteRootFunction der Klasse FunctionHierarchy (vgl. Abbildung 90).

In Schritt (1) wird die Gesamtfunktion (root:Function) ermittelt.

In Schritt (2) werden zu der Gesamtfunktion die Teilfunktionen ermittelt. Für jede gefundene Teilfunktion wird die Methode deleteChildFunction aufgerufen (siehe 8.3.1), wodurch rekursiv alle weiteren Teilfunktionen gelöscht werden. In diesem Schritt werden alle Funktionen mit Ausnahme der Gesamtfunktion gelöscht.

In Schritt (3) wird schließlich die Gesamtfunktion gelöscht.

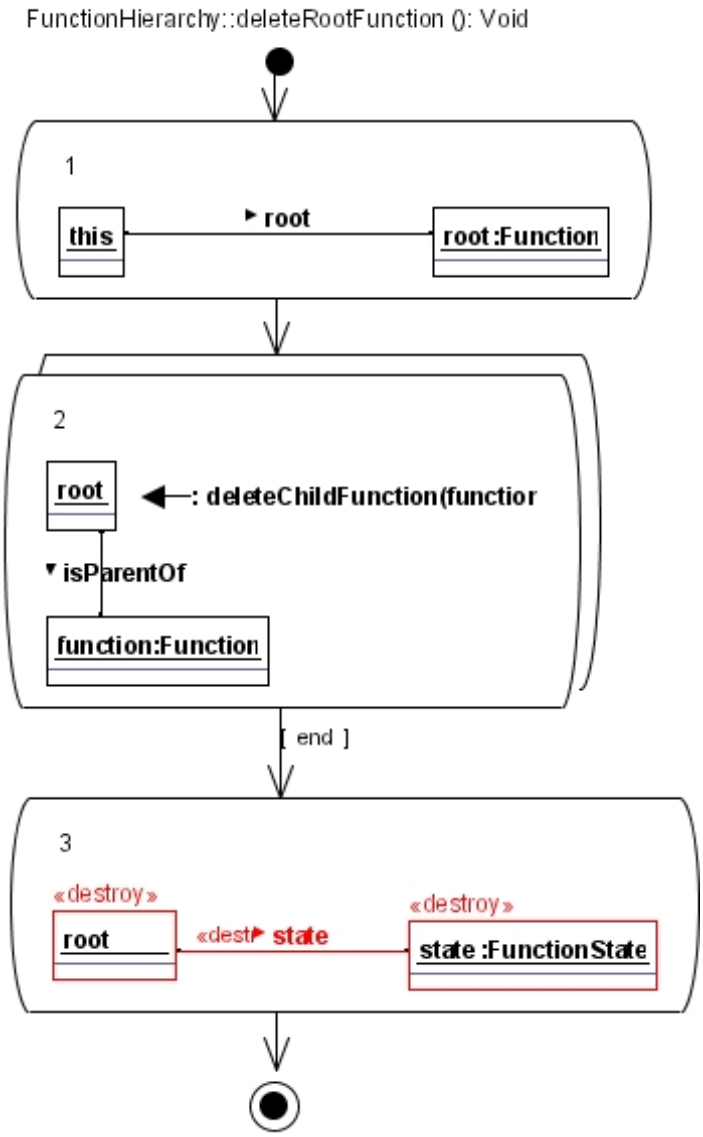


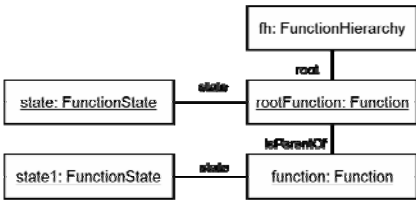
Abbildung 90: Löschen der Gesamtfunktion

Beispiel

Das folgende Beispiel (Abbildung 91) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel werden zuerst alle Teilfunktionen (function) inkl. der zugehörigen Statusobjekte (state1) gelöscht. Im Anschluss wird dann die Gesamtfunktion (rootFunction) und das zugehörige Statusobjekt (state) gelöscht.

Vor der Ausführung:



Nach der Ausführung:

Abbildung 91: Anwenden der Graphtransformationsregel "deleteRootFunction"

8.4 Verschieben von Funktionen

Neben dem Anlegen und Löschen einer Funktion kann es vorkommen, dass eine Funktion innerhalb der Funktionsstruktur verschoben werden muss. Dies kommt vor, wenn der Entwickler sich entschließt eine andere Strukturierung der Funktionshierarchie vorzunehmen. Dabei ist darauf zu achten, dass bereits zugewiesene Systemelemente inkl. ihrer Hilfsfunktionen nicht gelöscht werden. Aus diesem Grund ist ein Löschen und anschließendes Anlegen der Funktion nicht möglich.

Folgende Aktionen müssen beim Verschieben durchgeführt werden.

(1) Löschen der alten Verbindung und Hinzufügen einer neuen Verbindung

(2) Setzen des Status

Formale Beschreibung zum Verschieben einer Funktion

Das Verschieben einer Funktion wird mit Hilfe der Methode moveFunction durchgeführt. Die Dabei übergebene Funktion (newParentFunction) stellt die neue, übergeordnete Funktion (Vaterfunktion) dar.

In Schritt (1) in Abbildung 92 wird zuerst die Verbindung zur Vaterfunktion (oldParentFunction:Function) gelöscht und dann die neue Verbindung zur neuen Vaterfunktion (newParentFunction) hergestellt.

In Schritt (2) wird der Status für die alte und für die neue Vaterfunktion neu gesetzt, indem die Methode checkState() aufgerufen wird.

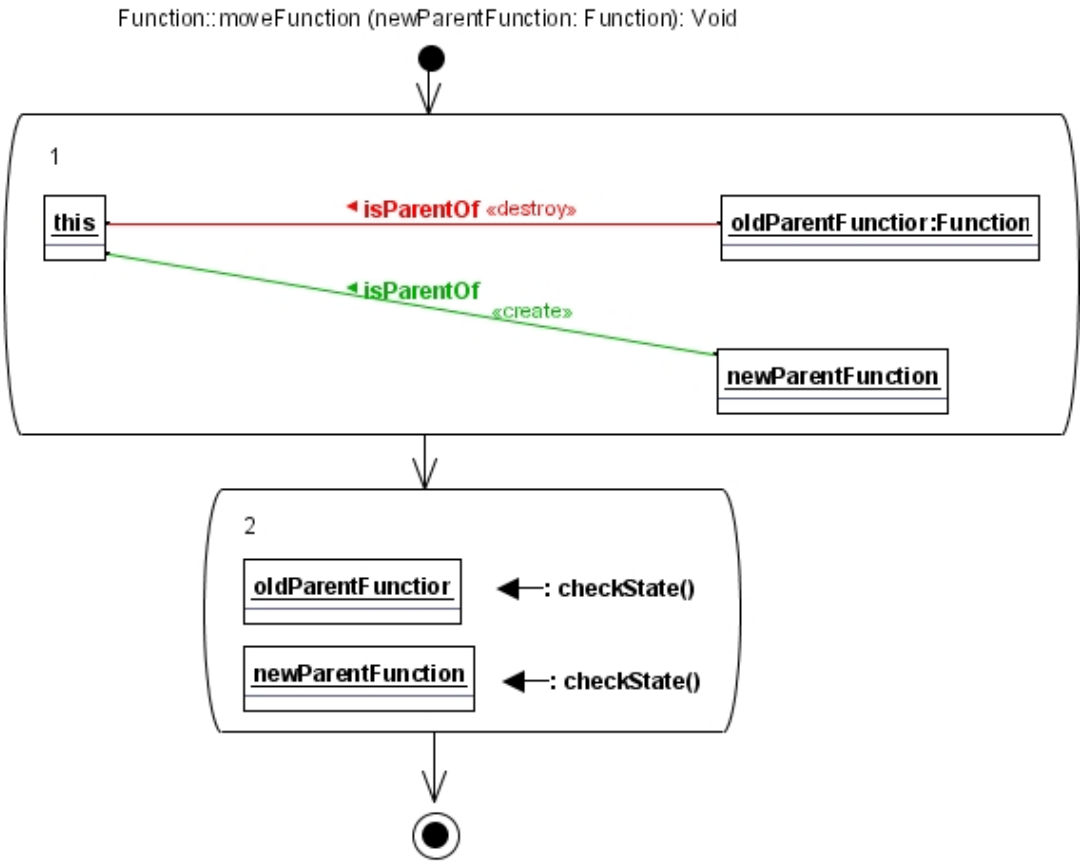


Abbildung 92: Verschieben einer Funktion innerhalb der Funktionshierachie

Beispiel

Das folgende Beispiel (Abbildung 93) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

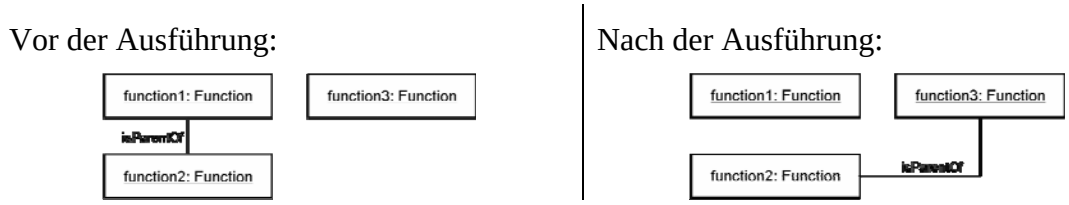


Abbildung 93: Anwenden der Graphtransformationsregel "moveFunction"

8.5 Anlegen von Systemelementen

Nach der Beschreibung der Operationen zur Manipulation der Funktionshierarchie werden im folgenden die Operationen zur Manipulation der Systemstruktur beschrieben. Die erste Operation ist das Anlegen eines Systemelements.

Anlegen eines Systemelements in einer Systemstruktur bedeutet, dass der Systemstruktur ein Systemelement hinzugefügt wird. Hierfür gibt es drei verschiedene Möglichkeiten. Die erste Möglichkeit ist die Ermittlung von Systemelementen mit Hilfe des in Kapitel sieben beschriebenen Suchalgorithmus. Die zweite Möglichkeit ist die manuelle Auswahl eines

Systemelements aus der Systemelementbibliothek und die dritte Möglichkeit ist das Anlegen eines neuen Systemelements innerhalb der Systemstruktur[31].

Sobald einer Systemstruktur ein Systemelement zugewiesen wird, muss dieses Systemelement einer Funktion (vgl. Konsistenzbedingung 1) zugewiesen werden. Wird keine konkrete Funktion angegeben, dann wird das Systemelement vorerst der Gesamtfunktion zugewiesen. Diese Zuweisung wird typischerweise nur von temporärer Dauer sein, da das Systemelement wahrscheinlich zur Realisierung einer Teilfunktion benötigt wird.

Die Zuweisung zur Gesamtfunktion hat den Zweck, dass bereits auf mögliche Hilfsfunktionen hingewiesen werden kann. Der Entwickler erhält direkt nach der Auswahl aus der Bibliothek ein entsprechendes Feedback, ob das gewählte Systemelement weitere Funktionen und damit weitere Systemelemente benötigt.

Bei der Anlage eines neuen Systemelements in der Systemstruktur wird dieses Systemelement ebenfalls der Gesamtfunktion zugewiesen. Hier können natürlich keine Hilfsfunktionen angegeben werden.

Die durchzuführenden Aktionen sind:

(1) Systemelement zur Systemstruktur hinzufügen

(2) Systemelement einer Funktion zuweisen

Formale Beschreibung zum Anlegen eines Systemelements

Das Anlegen neuer Systemelemente erfolgt mit Hilfe der Methode addSystemelement der Klasse Systemstructure (vgl. Abbildung 94). Übergeben werden dabei ein Objekt der Klasse Systemelement und ein Objekt der Klasse Function.

Zu Beginn wird geprüft, ob ein Objekt der Klasse Systemelement übergeben wurde. War dies der Fall (se != null), dann wird in Schritt (1) dieses Systemelement (se) mit Hilfe der entsprechenden <<create>> Anweisung mit der Systemstruktur (this) verknüpft.

Wurde kein Systemelement übergeben, dann wird ein neues Systemelement angelegt. Dies geschieht in Schritt (2). Hier wird zuerst ein neues Systemelement erzeugt (se:Systemelement) und dann mit Hilfe der <<create>> elements Anweisung mit der Systemstruktur (this) verknüpft.

Als nächstes wird überprüft, ob eine Funktion übergeben wurde, der das Systemelement zugewiesen werden soll. Wurde keine Funktion überwiesen (function == null), dann muss das Systemelement der Gesamtfunktion zugewiesen werden. Dies ist Notwendig, damit die Konsistenzbedingungen 1 und 2 eingehalten werden. In Schritt (3) wird zunächst ausgehend von der aktuellen Systemstruktur (this) zuerst die zugehörige Funktionshierarchie (functionHierarchy:FunctionHierarchy) ermittelt und von dort aus die Gesamtfunktion (root:Function). Dieser wird dann in Schritt (4) mit Hilfe der Methode assignSystemelement (vgl. 8.6) das übergebene bzw. neu angelegte Systemelement zugewiesen.

Wurde der Methode eine Funktion übergeben, dann wird dieser Funktion in Schritt (5) Hilfe der Methode assignSystemelement das übergebene bzw. neu angelegte Systemelement zugewiesen.

[31] Das Anlegen neuer Systemelemente innerhalb der Systemstruktur hat keinen Einfluss auf die Systemelementbibliothek. Eine automatische Integration in die Bibliothek ist nicht vorgesehen.

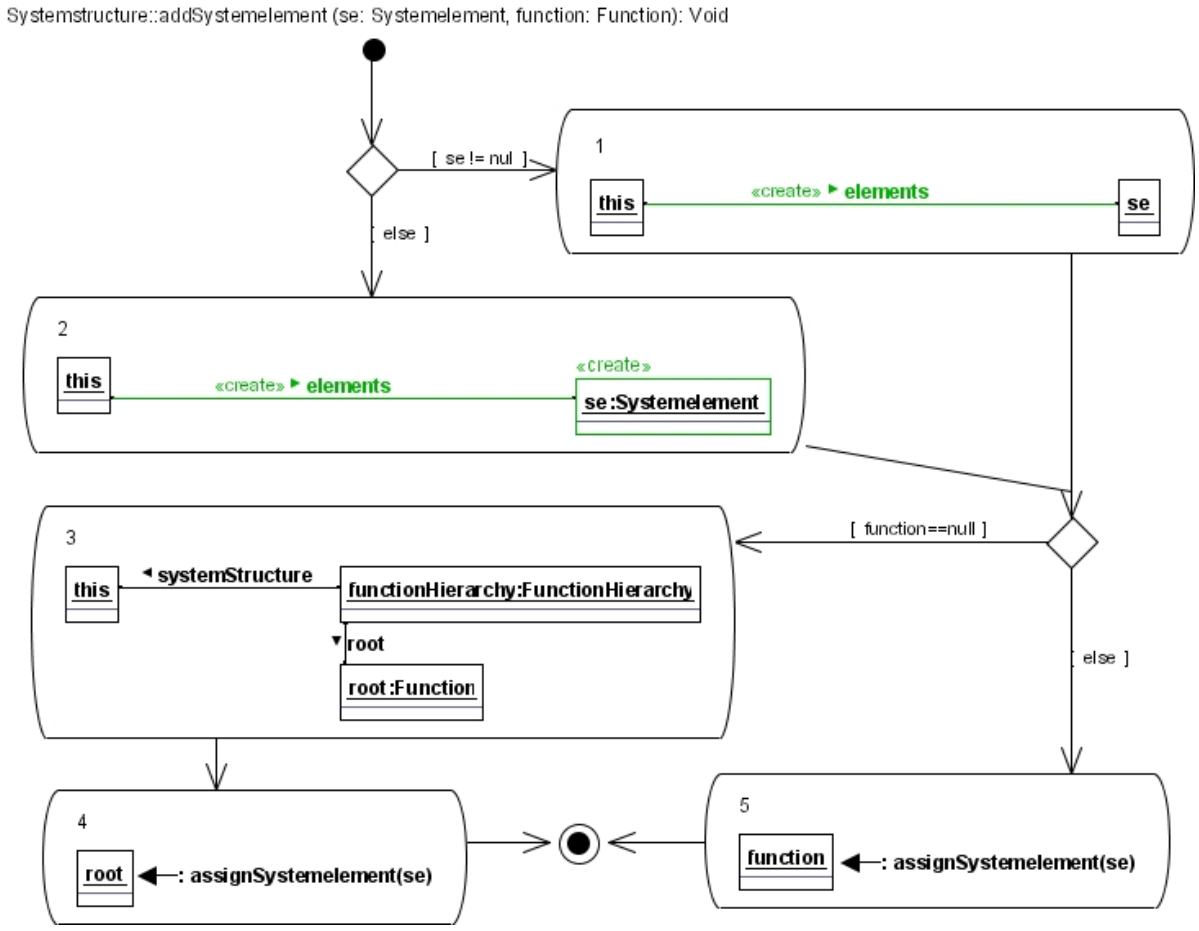


Abbildung 94: Anlegen von Systemelementen

Beispiel

Das folgende Beispiel (Abbildung 95) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In dem Beispiel wird ein neues Systemelement (se1) erzeugt und mit der Systemstruktur (ss) und der übergebenen Funktion (function1) verlinkt. Dies beschreibt die Ausführungsreihenfolge (2) und (5) der StoryPattern in Abbildung 94.

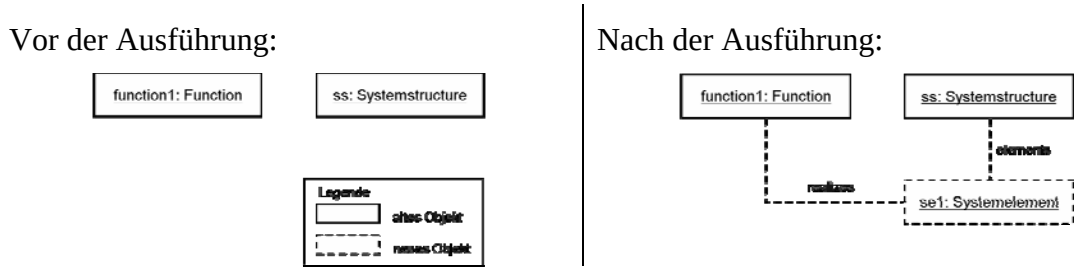


Abbildung 95: Anwenden der Graphtransformationsregel "addSystemelement"

8.6 Zuweisen von Systemelementen zu Funktionen

Unterschiedliche Systemelemente haben unterschiedliche Fähigkeiten, die sie in die Lage versetzen, bestimmte Funktionen zu erfüllen. Abhängig von diesen Fähigkeiten ist die Anzahl der zugewiesenen Systemelemente. Einige Funktionen werden bereits durch ein einzelnes Systemelement vollständig realisiert, andere benötigen mehrere Systemelemente.

Unter einer *Zuweisung* wird daher, im Rahmen dieser Arbeit, die Zuordnung eines Systemelements zu einer Funktion verstanden. Ist die Menge der zugewiesenen Systemelemente ausreichend, um die Funktion vollständig zu erfüllen, spricht man von einer *realisierten* Funktion. Dies entspricht dem Status „realisiert“ aus Abschnitt 8.1.4.

Jeder Funktion der Funktionshierarchie können Systemelemente zugewiesen werden. Dies gilt sowohl für Funktionen auf unterster Hierarchieebene, als auch für die Gesamtfunktion. Zugewiesen werden können sowohl einzelne, als auch mehrere Systemelemente, wobei es sich sowohl um hierarchische, als auch um atomare Systemelemente handeln kann.

Sobald jede Funktion einer Funktionshierarchie von Systemelementen realisiert wird, sind alle Anforderungen erfüllt und das mechatronische System ist bzgl. der einzusetzenden Systemelemente vollständig spezifiziert.

Die Zuweisungen der Systemelemente zu den Funktionen erfolgt durch den Entwickler. Nur er ist in der Lage zu entscheiden, welche Funktion durch welche Systemelemente realisiert werden können. Dies gilt auch für die in Kapitel 7 beschriebene Suche nach Systemelementen. Die auf diese Weise automatisch ermittelten Systemelemente werden dem Entwickler präsentiert und er entscheidet dann, welches Systemelement zugewiesen wird.

Durch das Zuweisen eines Systemelements zu einer Funktion wird die Konsistenzbedingung 1 aus 8.1.1 sicher gestellt. Insgesamt gibt es folgende Aktionen, die bei der Zuweisung von Systemelementen durchgeführt werden müssen:

(1) Feststellen, ob die Funktion eine Hilfsfunktion des Systemelements ist

(2) Zuweisen des Systemelements zur Funktion

(3) Hinzufügen der Hilfsfunktionen des Systemelements zur Funktionshierarchie

(4) Festlegen des Status

Formale Beschreibung der Zuweisung

Die Zuweisung eines Systemelements zu einer Funktion erfolgt mit Hilfe der Methode assignSystemelement der Klasse Function. In Schritt (1) in Abbildung 96 wird überprüft, ob die Funktion (this) nicht eine Hilfsfunktion (helpFunction-Link) des übergebenen Systemelements (se) ist. Dies ist nicht zulässig, da sonst ein Systemelement seiner eigenen Hilfsfunktionen zugewiesen würde und eine Schleife entsteht.

Handelt es sich bei der Funktion nicht um eine Hilfsfunktion des Systemelements, dann wird das Systemelement in Schritt (2) mit Hilfe der Assoziation realizes mit der Funktion verlinkt.

In Schritt (3) wird überprüft, ob das Systemelement von weiteren Hilfsfunktionen abhängig ist. Dies wird durch den Aufruf der Methode addHelpfunctions erreicht (vgl. 8.7). Falls Hilfsfunktionen vorhanden sind, dann wurden diese, nach Beendigung der Methode addHelpfunctions, unterhalb der Funktion (this) als Teilfunktionen in die Funktionshierarchie integriert.

In Schritt (4) wird der Status der Funktion mit Hilfe der Methode checkState (vgl. 8.13) überprüft und gegebenenfalls neu gesetzt.

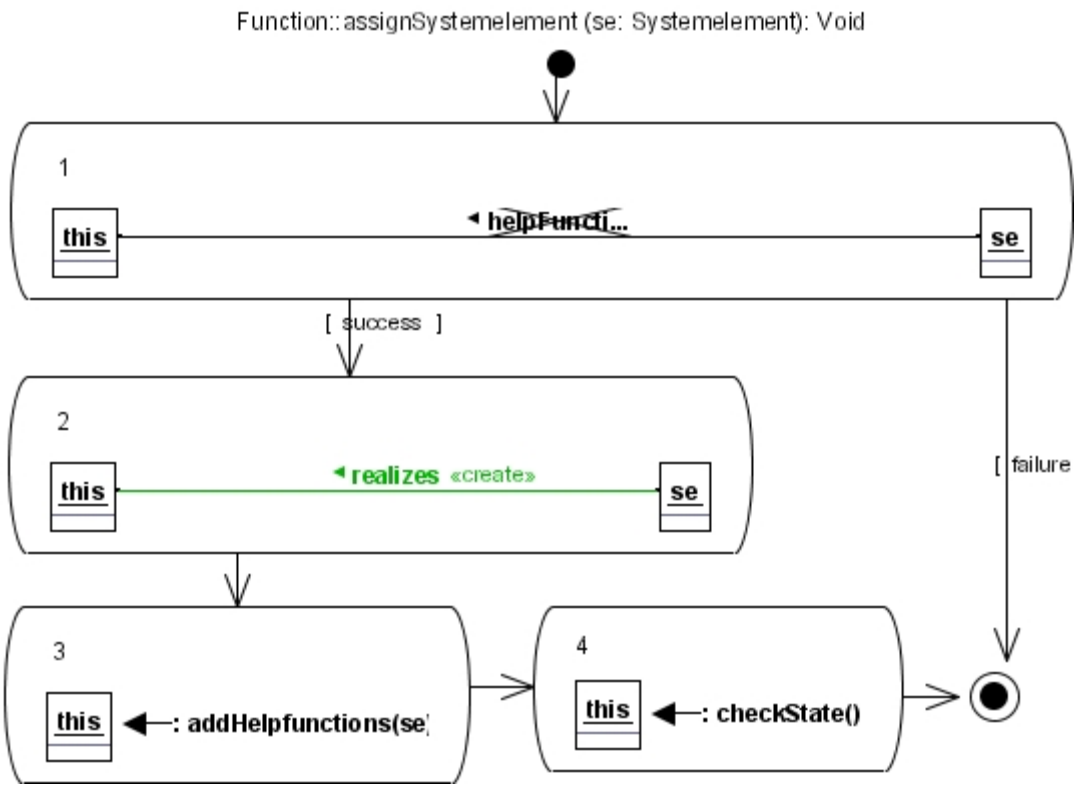


Abbildung 96: Zuweisen von Systemelementen zu Funktionen

Beispiel

Das folgende Beispiel (Abbildung 97) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

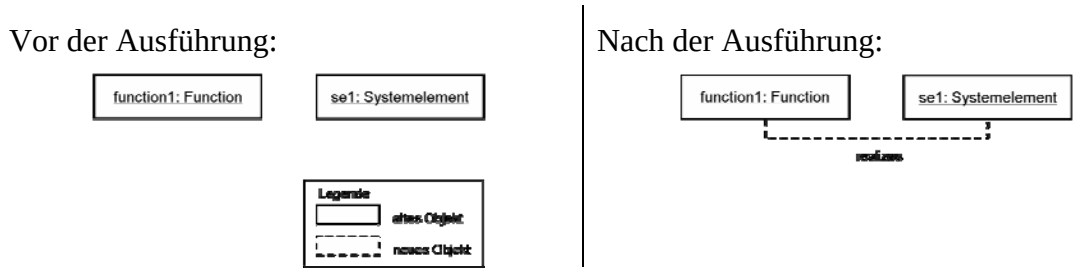


Abbildung 97: Anwenden der Graphtransformationsregel "assignSystemelement"

8.7 Ergänzen von Hilfsfunktionen

Nachdem ein Systemelement in die Systemstruktur aufgenommen wurde, muss geprüft werden, ob dadurch zusätzliche Funktionen (Hilfsfunktionen) zur Funktionshierarchie hinzukommen (vgl. Konsistenzbedingung 2).

In Kapitel 6.5 wurde gezeigt, dass ein Systemelement von anderen Funktionen abhängig sein kann, d.h. das es zusätzliche Funktionen und damit zusätzliche Systemelemente benötigt, um korrekt zu funktionieren. Konkret bedeutet das, dass die Funktionshierarchie um diese Hilfsfunktionen ergänzt werden muss, wobei diese Hilfsfunktionen an der richtigen Stelle in der Funktionshierarchie platziert werden müssen. Es gibt zwei verschiedene Stellen, an denen die Hilfsfunktionen platziert werden können.

(1) **Unterhalb einer Teilfunktion:** Wurde das Systemelement bereits einer Funktion zugewiesen, dann werden seine Hilfsfunktionen unterhalb der Funktion platziert, der er zugewiesen wurde. Wurde das Systemelement mehreren Funktionen zugewiesen, dann werden seine Hilfsfunktionen bei all diesen Funktionen als Teilfunktionen platziert.

(2) **Unterhalb der Gesamtfunktion:** Wurde das Systemelement noch keiner Funktion zugewiesen, dann wird das Systemelement automatisch der Gesamtfunktion zugewiesen (vgl. Abschnitt 8.6) und die Hilfsfunktionen unterhalb der Gesamtfunktion in die Funktionshierarchie integriert.

In Abbildung 98 wurde die Hilfsfunktion 1 unterhalb der Gesamtfunktion eingefügt, da das zugehörige Systemelement (nicht abgebildet), dass diese Hilfsfunktion benötigt, noch keiner Funktion zugewiesen wurde.

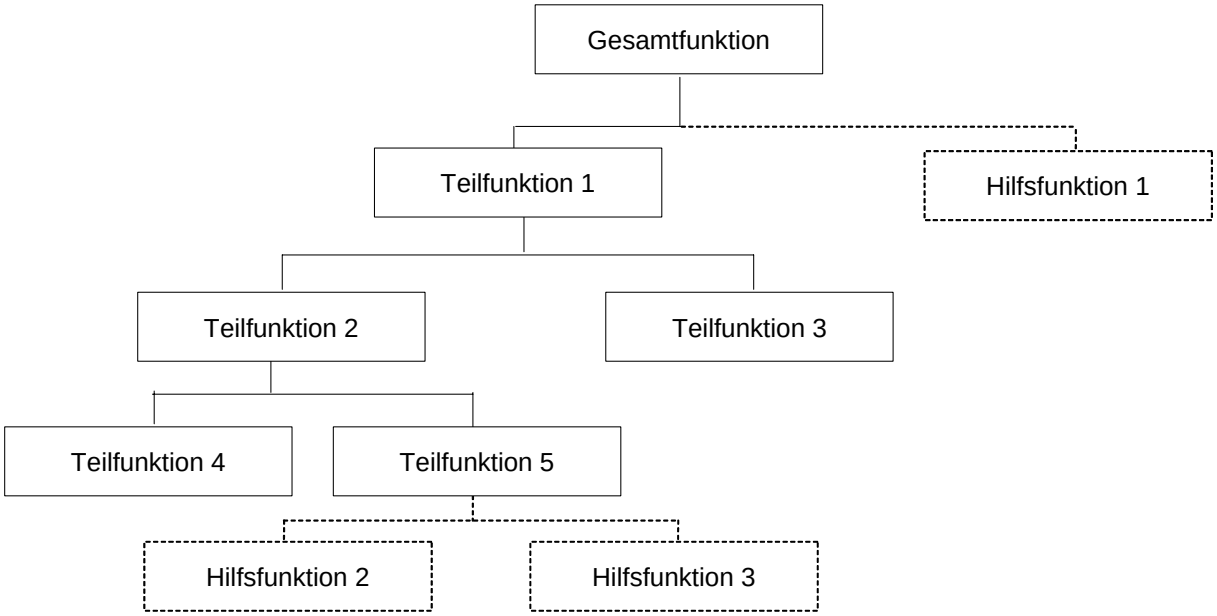


Abbildung 98: Integration von Teilfunktionen

Die Hilfsfunktionen 2 und 3 wurden unterhalb der Teilfunktion 5 platziert, da das Systemelement (nicht abgebildet), das diese Hilfsfunktionen mitbringt, der Teilfunktion 5 zugewiesen wurden.

Um Hilfsfunktionen zu ergänzen müssen folgende Aktionen durchgeführt werden:

(1) Ermitteln aller Hilfsfunktionen des Systemelements

(2) Erzeugen neuer Funktionen für die Funktionshierarchie mit den Charakteristiken der Hilfsfunktionen

(3) Hinzufügen der neuen Funktionen zur Funktionshierarchie

Formale Beschreibung der Integration der Hilfsfunktionen

Das Hinzufügen der Hilfsfunktionen erfolgt mit Hilfe der Methode addHelpfunctions. Die Methode wird auf der Funktion aufgerufen, an die die Hilfsfunktionen des übergebenen Systemelements angehängt werden sollen.

In Schritt (1) in Abbildung 99 wird zunächst geprüft, ob das Systemelement überhaupt Hilfsfunktionen besitzt. Dies lässt sich anhand des requires-Links ermitteln. Für jede gefundene Hilfsfunktion muss dann eine neue Funktion erstellt und in die Funktionshierarchie integriert werden.

Zu Schritt (2) gelangt man, wenn mindestens eine Hilfsfunktion identifiziert wurde. In diesem Schritt wird eine neue Funktion (helpFunction:Function) angelegt und mit dem Attribut helpFunction = true initialisiert. Als nächstes wird das Verb der Hilfsfunktion ermittelt. Dies wird mit Hilfe der Assoziation selectedVerb zwischen den Objekten verb:verbName und sef erreicht. Für das gefundene Verb wird ein Link (<<create>> functionVerb) zur neu angelegten Hilfsfunktion erstellt.

Im Anschluss daran werden in Schritt (3) alle Input-Substantive ermittelt. Hierzu werden alle Links der Assoziation selectedOutput ermittelt und jeweils ein functionOutput-Link zwischen der helpFunction und dem subject erzeugt.

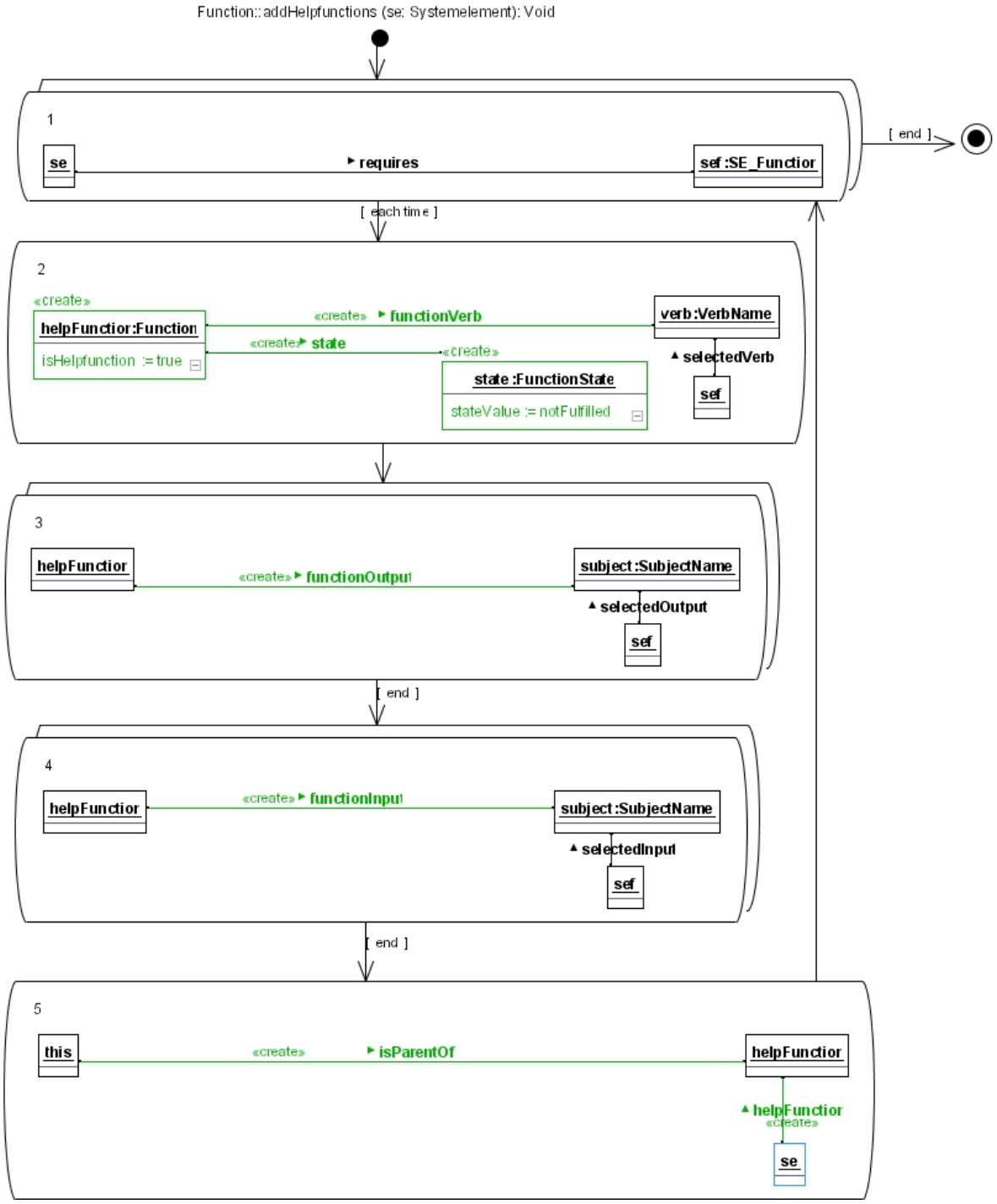


Abbildung 99: Hilfsfunktionen zur Funktionshierarchie hinzufügen

Schritt (4) gleicht dem Schritt (3), allerdings werden hier die InputSubstantive ermittelt und entsprechend verlinkt.

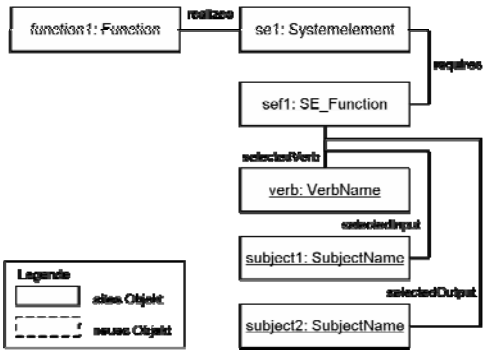
In Schritt (5) erfolgt dann die eigentliche Integration der neuen Funktion in die Funktionshierarchie. Zuerst wird die neue Funktion (helpFunction) als Kindknoten (Link isParentOf) mit der aktuellen Funktion (this) verlinkt. Das bedeutet, dass die neue Funktion nun eine Teilfunktion der ursprünglichen Funktion ist. Schließlich wird dann mit Hilfe des helpFunction-Link zwischen der Hilfsfunktion und dem Systemelement festgehalten, dass diese Hilfsfunktion von genau diesem Systemelement stammt. Dies ist für ein späteres „umhängen“ der Hilfsfunktionen wichtig.

Beispiel

Das folgende Beispiel (Abbildung 100) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel hat das Systemelement (se1) nur eine Hilfsfunktion (sef1) mit einem Verb (verb), einem Input-Substantiv (subject1) und einem Output-Substantiv (subject2).

Vor der Ausführung:



Nach der Ausführung:

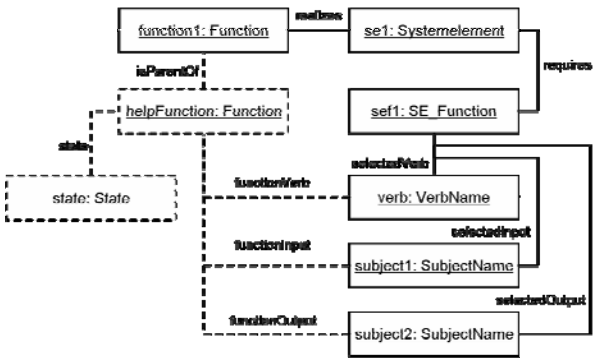


Abbildung 100: Anwenden der Graphtransformationsregel "addHelpfunctions"

8.8 Löschen von Zuweisungen

Während der Modellierung der Funktionshierarchie und Systemstruktur kann es vorkommen, dass neue Ideen in die Modellierung einfließen, oder neue Randbedingungen oder Anforderungen bekannt werden. Dies kann dann zur Folge haben, dass andere Systemelemente eingesetzt werden sollen als bisher. Dadurch wird es erforderlich, dass bereits getätigte Zuweisungen wieder Rückgängig gemacht werden können.

Wird die Zuweisung eines Systemelements zu einer Funktion aufgehoben, das Systemelement aber in der Systemstruktur belassen, dann muss das Systemelement aufgrund der Konsistenzbedingung 1, der Gesamtfunktion zugewiesen werden, wenn es keiner anderen Funktion mehr zugewiesen ist.

Beim Aufheben der Zuweisung muss dann darauf geachtet werden, dass die Hilfsfunktionen, die das Systemelement „mitgebracht“ hat, an der entsprechenden Stelle in der Funktionshierarchie gelöscht und gegebenenfalls unterhalb der Gesamtfunktion neu angeordnet werden (Konsistenzbedingung 2).

Folgende Aktionen müssen für das Löschen einer Zuweisung durchgeführt werden:

(1) Ermitteln aller Teilfunktionen und Feststellen, ob es sich um Hilfsfunktionen des Systemelements handelt

(2) Löschen bzw. verschieben vorhandener Hilfsfunktionen

(3) Löschen der Zuweisung

(4) Möglicherweise Zuweisung des Systemelements zur Gesamtfunktion

(5) Festlegen des Status

Formale Beschreibung für die Aufhebung einer Zuweisung

Das Aufheben einer Zuweisung wird mit Hilfe der Methode unassignSystemelement der Klasse Function durchgeführt. Das Systemelement, dessen Zuweisung aufgehoben werden soll, wird als Parameter übergeben.

In Schritt (1) in Abbildung 101 wird die Zuweisung des Systemelements (se) zur aktuellen Funktion (this) gelöscht. Dies ist durch das Schlüsselwort <<destroy>> gekennzeichnet.

In Schritt (2) wird geprüft, ob das Systemelement noch anderen Funktionen zugewiesen wurde (realizes). Ist dies der Fall, dann müssen alle Hilfsfunktionen, die von dem Systemelement „mitgebracht“ wurden und sich noch unterhalb der Funktion befinden, gelöscht werden. Dies erfolgt in Schritt (3). Wurde das Systemelement keiner weiteren Funktion zugewiesen, dass muss es, gemäß Konsistenzbedingung 2, der Gesamtfunktion zugewiesen werden. Dies erfolgt in Schritt (8).

In Schritt (3) werden zunächst alle Teilfunktionen (childFunction:Function) der Funktion (this) ermittelt. Für jede der ermittelten Funktionen wird dann in Schritt (4) geprüft, ob es sich um eine Hilfsfunktion des übergebenen Systemelements handelt. Dies geschieht mit Hilfe des helpFunction-Links zwischen den Objekten se und childFunction.

Ist die Funktion eine Hilfsfunktion des übergebenen Systemelements, dann wird in Schritt (5) der Link zwischen der Hilfsfunktion (childFunction) und der aktuellen Funktion (this) gelöscht.

Nachdem der Link gelöscht wurde, wird nun in Schritt (6) die Hilfsfunktion selber gelöscht. Dies erfolgt mit Hilfe der Methode deleteChildFunction (vgl. 8.3.1). In dieser Methode werden auch sämtliche Teilfunktionen gelöscht, die sich unterhalb der Hilfsfunktion befinden.

Nachdem alle Hilfsfunktionen gelöscht wurden, wird in Schritt (7) mit Hilfe der Methode checkState() der neue Status der aktuelle Funktion (this) bestimmt.

Schritt (8) wird ausgeführt, wenn das Systemelement keiner weiteren Funktion zugewiesen ist und somit der Gesamtfunktion zugewiesen werden muss. In diesem Schritt wird zunächst daher zunächst die Gesamtfunktion (root) ermittelt.

In Schritt (9) wird mit Hilfe der Methode moveAssignment(this, root) die vorhandene Zuweisung zur Gesamtfunktion verschoben (vgl. 8.9). Da bereits innerhalb dieser Methode der Status neu gesetzt wird, ist dies hier nicht erneut notwendig.

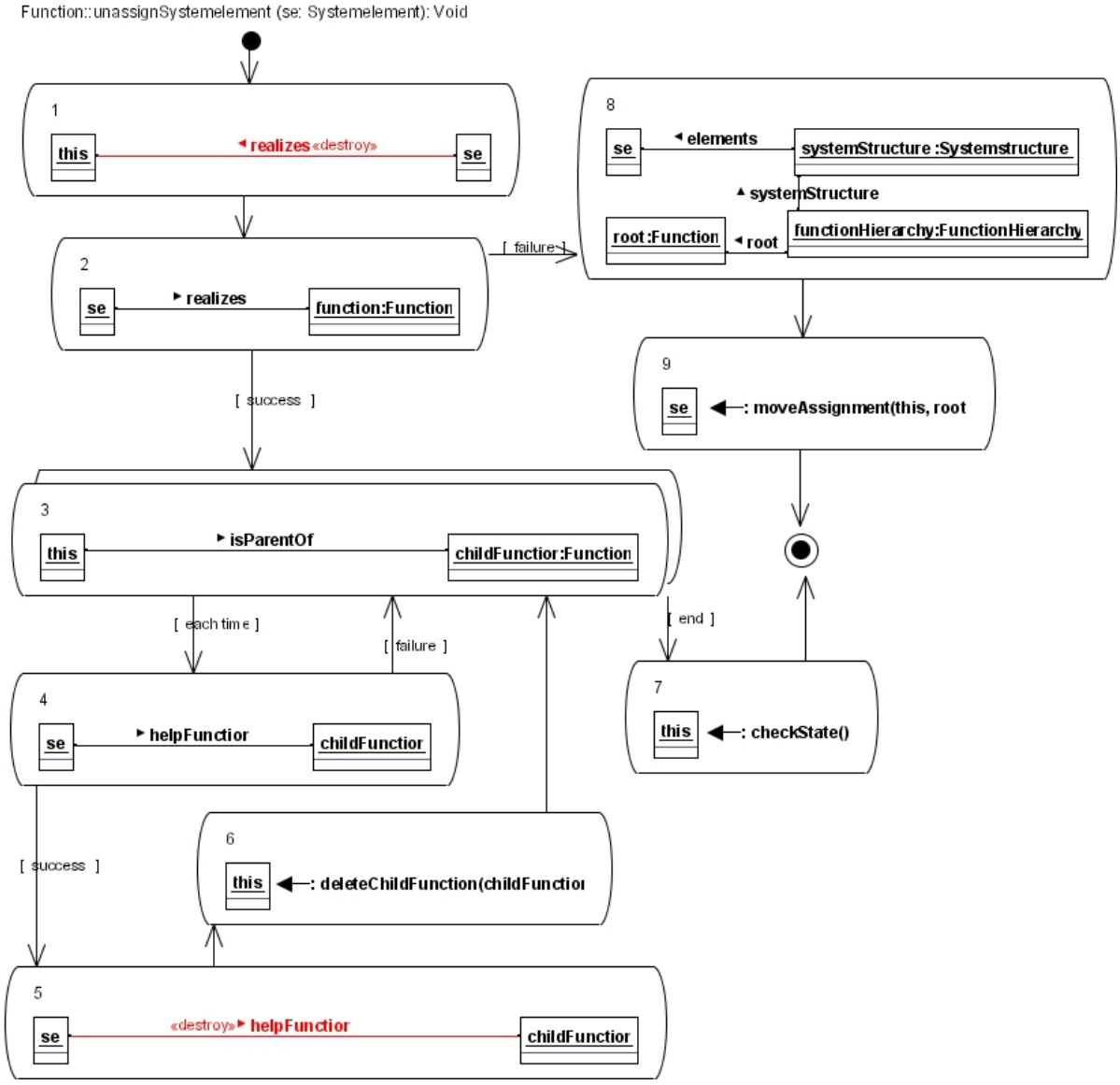


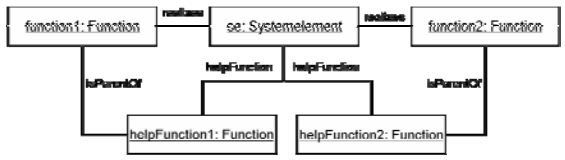
Abbildung 101: Aufheben einer Zuweisung

Beispiel

Das folgende Beispiel (Abbildung 102) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel wird die Zuweisung eines Systemelements (se) zu einer Funktion (function1) aufgehoben. Durch das Aufheben der Zuweisung wird die Hilfsfunktion (helpFunction1) überflüssig und kann gelöscht werden. Das Beispiel entspricht der Ausführung der Schritte (1) bis (7) aus Abbildung 101. Die Schritte (8) und (9) wären ausgeführt worden, wenn das Systemelement ausschließlich der einen Funktion zugewiesen wäre. Dann hätte das Systemelement der Gesamtfunktion zugewiesen werden müssen.

Vor der Ausführung:



Nach der Ausführung:

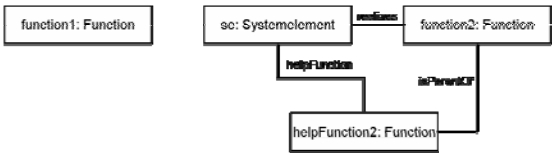


Abbildung 102: Anwenden der Graphtransformationsregel "unassignSystemelement"

8.9 Verschieben von Zuweisungen

Die Funktionsmodellierung unterliegt ständigen Änderungen. Neben dem Anlegen (vgl. 8.6) oder Aufheben einer Zuweisung (vgl. 8.8) kann es vorkommen, dass eine Zuweisung von der einen Funktion zur anderen verschoben werden soll.

In diesem Fall muss nicht nur die Zuweisung selbst, sondern auch die möglicherweise vorhandenen Hilfsfunktionen mit verschoben werden. Hierbei ist darauf zu achten, dass Zuweisungen, die bereits für die Hilfsfunktionen getätigt wurden, nicht verloren gehen. Aus diesem Grund ist es nicht möglich eine Zuweisung einfach zu löschen und bei einer anderen Funktion neu anzulegen.

Die durchzuführenden Aktionen sind:

(1) Verschieben aller Hilfsfunktionen

(2) Verschieben der Zuweisung

(3) Festlegen des Status

Formale Beschreibung für das Verschieben einer Zuweisung

Das Verschieben einer Zuweisung wird mit Hilfe der Funktion moveAssignment der Klasse Systemelement durchgeführt. Als Parameter werden sowohl die alte, als auch die neue Funktion übergeben. Die Übergabe der alten Funktion ist notwendig, da ein Systemelement auch mehreren Funktionen zugewiesen sein kann.

In Schritt (1) wird zunächst die Methode moveHelpfunctions aufgerufen. Diese Methode verschiebt die von dem Systemelement stammenden Hilfsfunktionen von der alten Funktion zur neuen (vgl. 8.10). Dabei bleiben alle Zuweisungen von Systemelementen zu den Hilfsfunktionen erhalten.

In Schritt (2) wird die eigentliche Zuweisung zur alten Funktion (oldFunction) gelöscht und der neuen Funktion (newFunction) zugewiesen.

In Schritt (3) wird sowohl auf der alten Funktion als auch auf der neuen Funktion die Methode checkState aufgerufen.

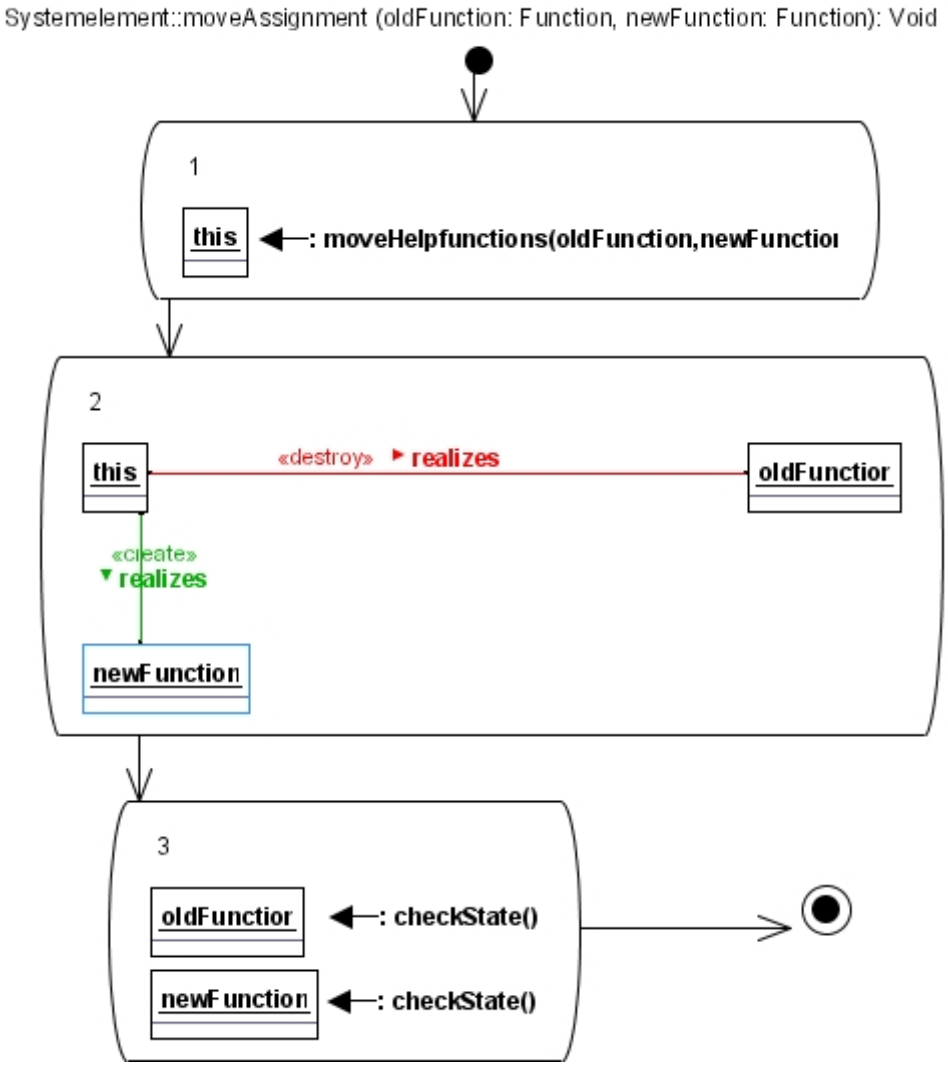


Abbildung 103: Zuweisungen verschieben

Beispiel

Das folgende Beispiel (Abbildung 104) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel wird die Zuweisung eines Systemelements (se) zu einer Funktion (function) aufgehoben und der Gesamtfunktion (root) zugewiesen. Dabei wird auch die Hilfsfunktion (helpFunction) zur Gesamtfunktion verschoben.

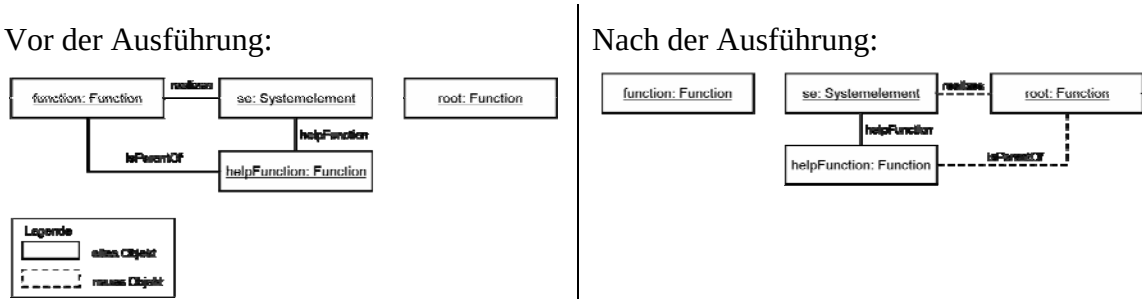


Abbildung 104: Anwenden der Graphtransformationsregel "moveAssignment"

8.10 Verschieben von Hilfsfunktionen

In Abschnitt 8.9 wurde gezeigt, wie eine Zuweisung verschoben wird. Dabei müssen auch alle Hilfsfunktionen verschoben werden, ohne dass hierfür bereits vorgenommene Zuweisungen verloren gehen.

Die durchzuführenden Aktionen sind:

(1) Ermitteln aller Teilfunktionen der alten Funktion

(2) Feststellen, ob eine Teilfunktion eine Hilfsfunktion ist

(3) Verschieben der Hilfsfunktion

Formale Beschreibung der Verschiebung von Hilfsfunktionen

Durchgeführt wird das Verschieben von Hilfsfunktionen mit Hilfe der Methode moveHelpfunctions der Klasse Systemelement. Als Parameter werden die alte und die neue Funktion übergeben.

In Schritt (1) werden die Teilfunktionen (helpFunction:Function) der alten Funktion (oldFunction) ermittelt und der Reihe nach durchgegangen (eachtime). In Schritt (2) wird dann geprüft, ob die jeweilige Teilfunktion (helpFunction) eine Hilfsfunktion des Systemelements (this) ist. Ist dies der Fall, dann muss die Teilfunktion verschoben werden.

In Schritt (3) wird dann der Link zwischen der alten Funktion und der Hilfsfunktion gelöscht (<<destroy>>) und ein neuer Link zur neuen Funktion erzeugt (<<create>>).

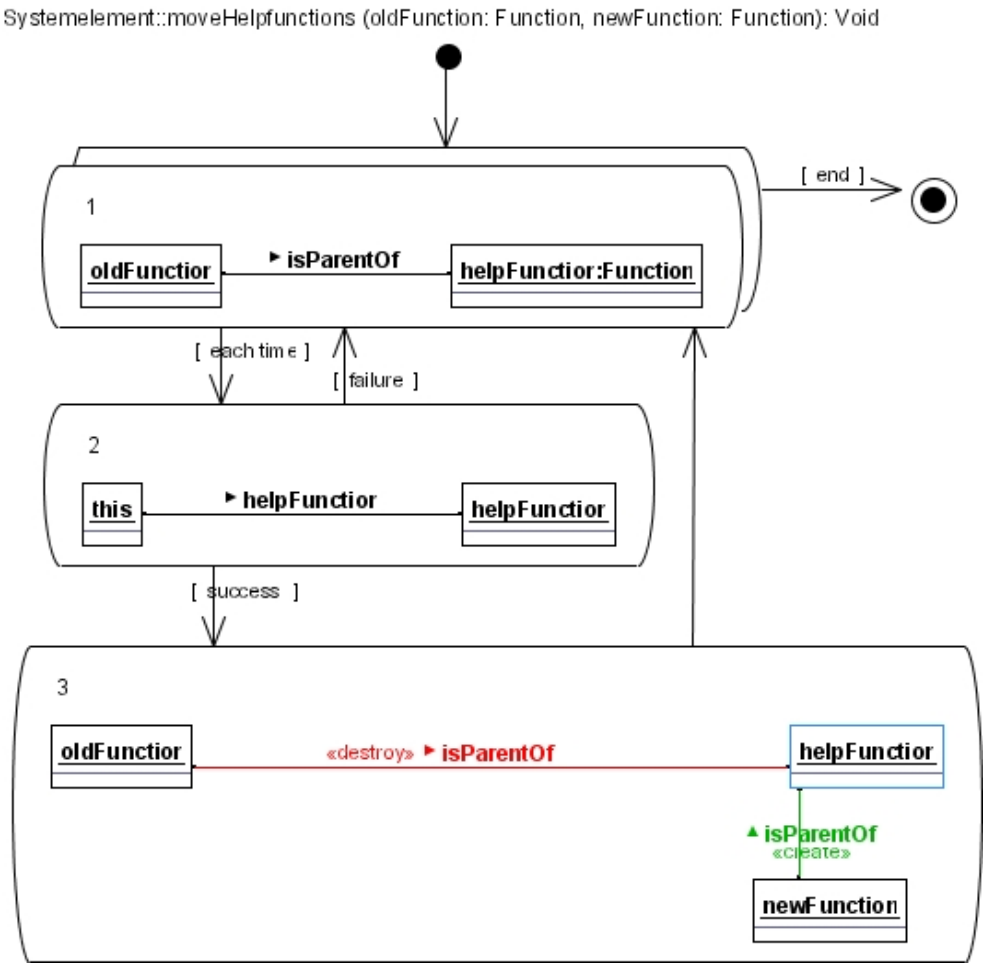


Abbildung 105: Verschieben von Hilfsfunktionen

Beispiel

Das folgende Beispiel (Abbildung 106) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

Dieses Beispiel gleicht dem Beispiel in Abschnitt 8.9. Die Hilfsfunktionen (helpFunction) eines Systemelement (se) von einer Funktion (function1) zur anderen (function2) verschoben.

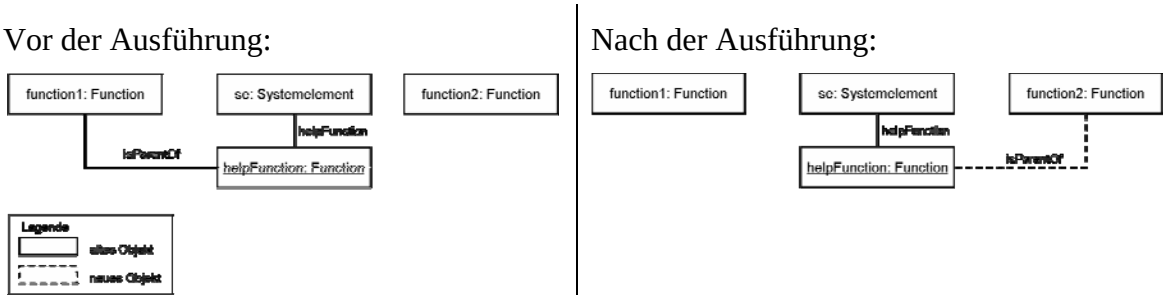


Abbildung 106: Anwenden der Graphtransformationsregel "moveHelpfunctions"

8.11 Löschen von Systemelementen

Die letzte Möglichkeit die Funktionshierarchie und Systemstruktur zu beeinflussen ist durch das Löschen eines Systemelements. Aufgrund der bereits erwähnten, ständigen Änderungen der Anforderungen, kann es sein, dass ein Systemelement nicht mehr benötigt wird oder der Entwickler sich für ein anderes entscheidet. Dieses Systemelement muss dann aus der Systemstruktur gelöscht werden.

Hierbei ist darauf zu achten, dass auch seine Hilfsfunktionen gelöscht werden (Konsistenzbedingung 3). Dies kann jedoch zur Folge haben, dass Systemelemente die diesen Hilfsfunktionen zugewiesen wurden, anderen Funktionen zugewiesen werden müssen (Konsistenzbedingung 1). Ebenfalls gelöscht werden sollen diese Systemelemente nicht, da sie evtl. zur Realisierung einer anderen Funktion verwendet werden sollen. Würden sie gelöscht, dann müssten später auch für ihre Hilfsfunktionen erneut Systemelemente ermittelt werden, obwohl dies u. U. bereits geschehen ist.

Aus diesem Grund werden Systemelemente, die Hilfsfunktionen zugewiesen wurden, nicht gelöscht, sondern der Gesamtfunktion zugeordnet. Somit obliegt es dem Entwickler diese Systemelemente im Nachhinein selbst zu löschen.

Beim Löschen eines Systemelements müssen folgende Aktionen durchgeführt werden:

(1) Löschen aller Zuweisungen

(2) Löschen aller Hilfsfunktionen

(3) Verschieben von Zuweisungen zur Gesamtfunktion

(4) Löschen des Systemelements

Formale Beschreibung des Löschens eines Systemelements

Durchgeführt wird das Löschen eines Systemelements mit Hilfe der Methode deleteSystemelement der Klasse Systemstructure. Hier stellt das übergebene Objekt der Klasse Systemelement das zu löschende Systemelement dar.

In Schritt (1) in Abbildung 107 werden alle Funktionen (function:Function) ermittelt, denen das Systemelement zugewiesen wurde. Für jede gefundene Funktion wird dann in Schritt (2)

die Methode unassignSystemelement aufgerufen, wodurch die jeweilige Zuweisung zur Funktion gelöscht wird. Darüber hinaus wird innerhalb dieser Methode (vgl. 8.8) auch die den Hilfsfunktionen zugewiesenen Systemelemente der Gesamtfunktion zugewiesen.

Nachdem alle Zuweisungen aufgehoben wurden, werden in Schritt (3) alle Hilfsfunktionen mit Hilfe der Methode deleteAllHelpfunctions gelöscht (vgl. 8.12). Zum Abschluss wird in Schritt (4) das Systemelement aus der Systemstruktur gelöscht.

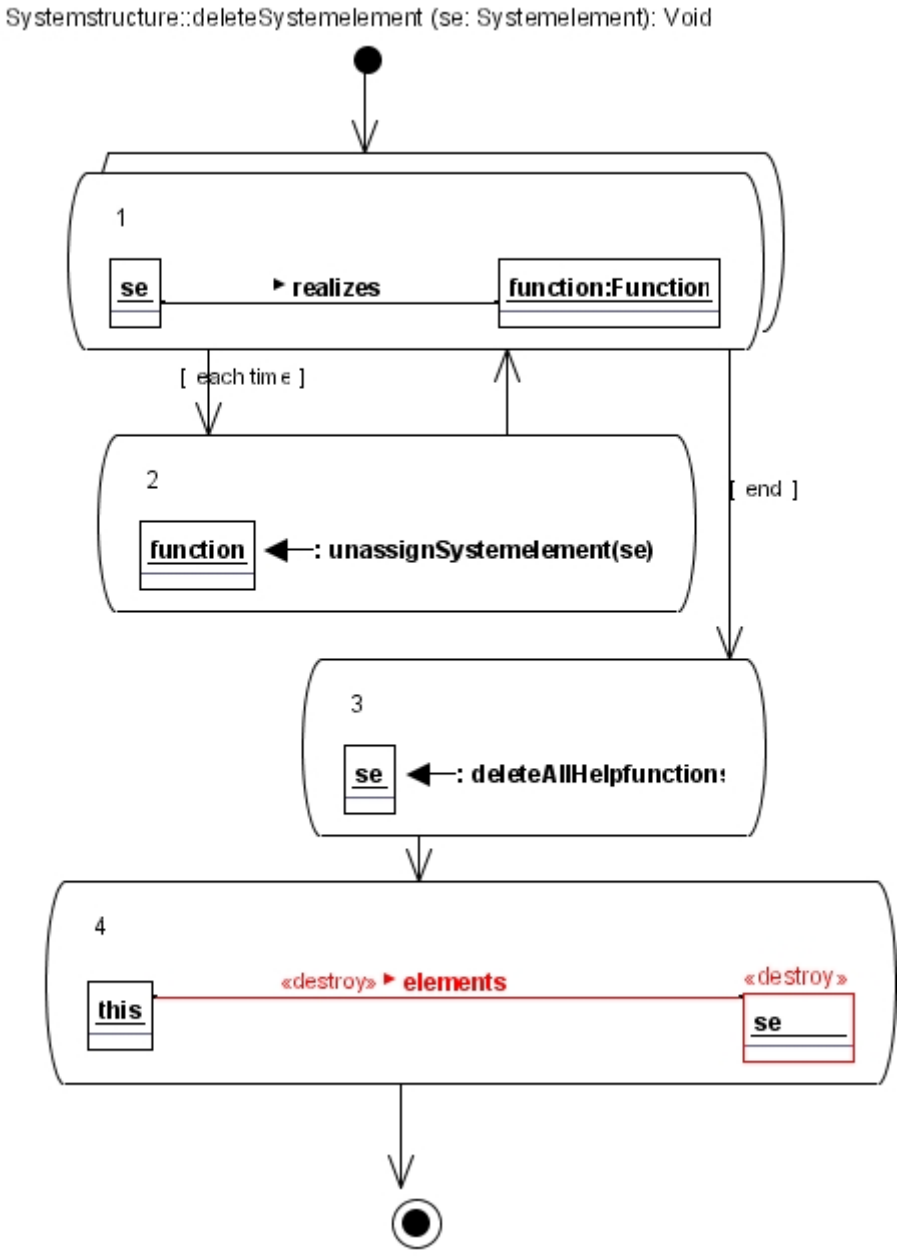


Abbildung 107: Löschen eines Systemelements

Beispiel

Das folgende Beispiel (Abbildung 108) zeigt die Ausführung der Methode bzw. der darin enthaltenen Graphtransformationsregeln. Die linke Seite zeigt den Objektgraphen vor der Ausführung und die rechte Seite zeigt den Objektgraphen nach der Ausführung.

In diesem Beispiel wird das Systemelement (se) und alle zugehörigen Hilfsfunktionen (helpFunction) gelöscht.

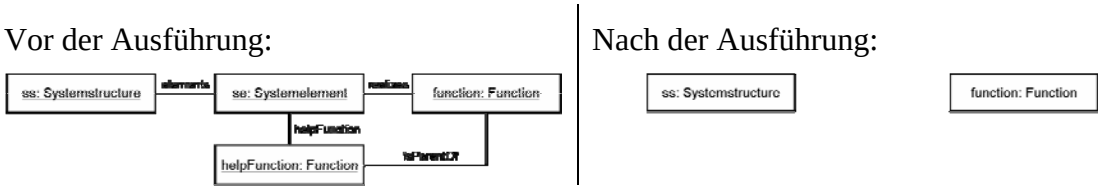


Abbildung 108: Anwenden der Graphtransformationsregel "deleteSystemelement"

8.12 Löschen von Hilfsfunktionen

Wird ein Systemelement aus der Systemstruktur gelöscht, dann müssen auch die durch ihn zur Funktionshierarchie hinzugekommenen Hilfsfunktionen gelöscht werden. Diese Hilfsfunktionen sind durch das Löschen des Systemelements unnötig geworden.

Diese Operation erfüllt die Konsistenzbedingung 3 aus Abschnitt 8.1.1.

Formale Beschreibung des Löschens der Hilfsfunktionen

Das Löschen der Hilfsfunktionen erfolgt mit Hilfe der Methode deleteAllHelpfunction der Klasse Systemelement. Alle von diesem Systemelement in der Funktionshierarchie enthaltenen Hilfsfunktionen werden gelöscht.

In Schritt (1) in Abbildung 109 werden alle Funktionen (helpFunction:Function) ermittelt die durch das Systemelement (this) in die Funktionshierarchie integriert wurden (helpFunction-Link). Jede dieser Funktionen inkl. der vorhandenen Links wird gelöscht (Schlüsselwort <<destroy>>).

Am Ende existiert innerhalb der Funktionshierarchie keine Hilfsfunktion mehr, die aufgrund des Systemelements in die Funktionshierarchie integriert wurde.

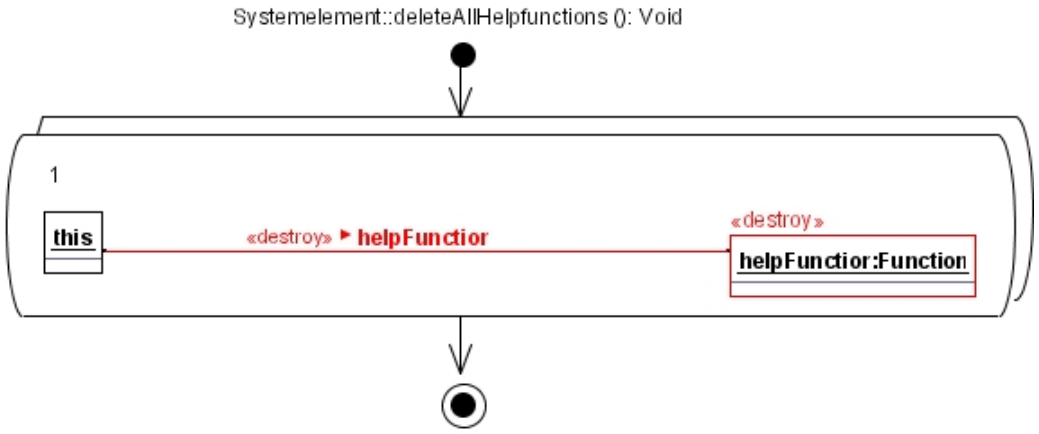


Abbildung 109: Löschen der Hilfsfunktionen

8.13 Statusvergabe

Die Vergabe eines Status für die Funktionen der Funktionshierarchie trägt dazu bei den aktuellen Entwurfstand zu verfolgen. Wie in Abschnitt 8.1.4 bereits beschrieben, erhält jede Funktion einen von drei Stati („nicht realisiert“, „teilweise realisiert“ und „realisiert“). Die Vergabe dieser Stati kann entweder manuell oder automatisch erfolgen.

Die Vergabe eines Status erfüllt die Konsistenzbedingung 4 aus 8.1.1.

8.13.1 Manuelle Statusvergabe

Bei der manuellen Statusvergabe entscheidet der Entwickler, welchen Status eine Funktion annehmen soll. Legt der Entwickler den Status einer Funktion fest, dann wird anschließend die automatische Statusvergabe aktiviert, um zu überprüfen, ob die Stati der übergeordneten Funktionen ebenfalls geändert werden müssen.

Der Status einer Funktion, der einmal manuell geändert wurde, kann nicht durch die automatische Statusvergabe erneut geändert werden.

Obwohl hauptsächlich die automatische Statusvergabe genutzt werden soll, gibt es eine Situation in der die manuelle Statusvergabe zwingend erforderlich ist. Diese Situation entsteht bei Teilfunktionen, die selbst keine Teilfunktionen mehr besitzen (Blätter der Funktionshierarchie), ihnen aber bereits Systemelemente zugewiesen wurden (vgl. 8.1.4). In diesem Fall lässt sich nicht automatisch bestimmen, ob die bereits zugewiesenen Systemelemente ausreichen, um die Funktion zu realisieren. Dies muss durch den Entwickler erfolgen.

Formale Beschreibung der manuellen Statusvergabe

Das manuelle setzen des Status erfolgt mit Hilfe der Methode setStateManually der Klasse Function. Übergeben werden der Methode der zusetzende Status (state:Integer).

In Schritt (1) in Abbildung 110 wird das zur Funktion gehörende Statusobjekt (functionState:FunctionState) ermittelt.

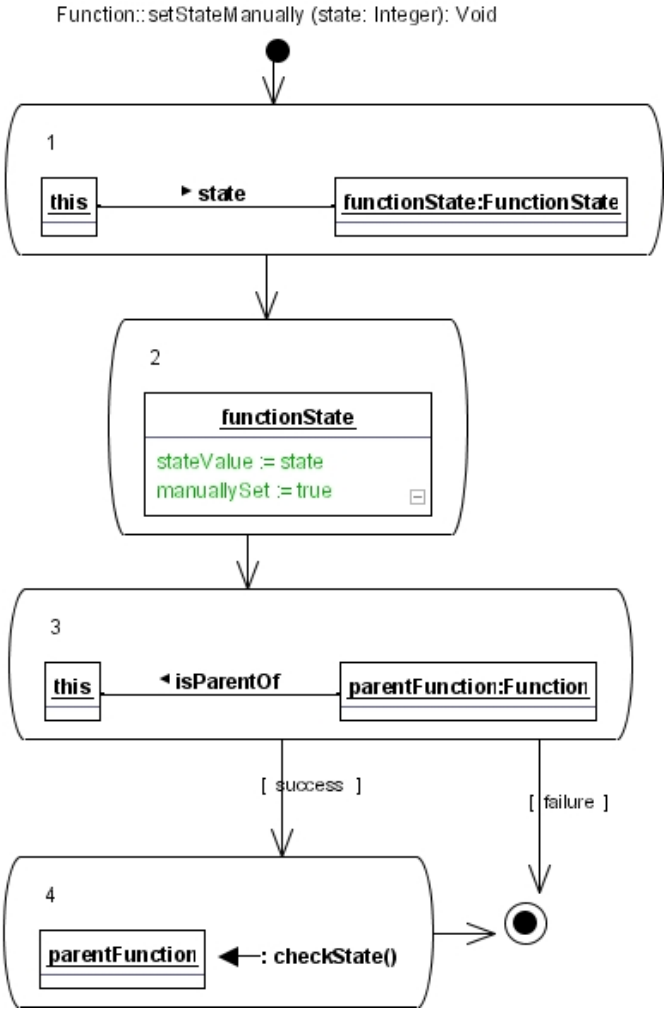


Abbildung 110: Manuelle Statusvergabe

Auf dem Objekt wird dann in Schritt (2) zunächst das Attribute stateValue auf den der Methode übergebenen Status (state) gesetzt. Zusätzlich wird das Attribut manuallySet auf true gesetzt um festzuhalten, dass dieser Status vom Entwickler gesetzt wurde und somit nicht mehr automatisch geändert werden darf.

In Schritt (3) wird der „Vaterknoten“, d.h. die übergeordnete Funktion gesucht. Dieser Funktion wird dann in Schritt (4) durch Aufruf der Methode checkState() mitgeteilt, dass sie ihren eigenen Status überprüfen soll.

8.13.2 Automatische Statusvergabe

Neben der manuellen Statusvergabe gibt es noch die automatische Statusvergabe. Sie wird benötigt, um nach Modifikationen der Funktionshierarchie oder der Systemstruktur neue Stati betroffener Funktionen festzulegen Damit dies automatisch möglich ist, wurden verschiedene Regeln aufgestellt die die Vergabe der Stati festlegen. Diese Regeln wurden umgangssprachlich bereits in 8.1.4 erläutert und werden im folgenden formal spezifiziert.

Die Vorgehensweise der automatischen Statusvergabe zeichnet sich dadurch aus, dass ausgehend von einer Funktion der Funktionshierarchie, der Status dieser Funktion und aller übergeordneten Funktionen bis hin zur Gesamtfunktion geprüft und gegebenenfalls neu gesetzt werden. Die Stati untergeordneter Funktionen bleiben unverändert.

Formale Beschreibung der automatischen Statusvergabe

Die automatische Statusvergabe wird mit Hilfe der Methode checkState der Klasse Function durchgeführt. Übergabeparameter hat die Methode nicht.

In Schritt (1) wird zunächst das zur Funktion gehörende Statusobjekt (functionState:FunctionState) ermittelt. Im Anschluss daran wird geprüft, ob der aktuelle Status manuell gesetzt wurde. Ist dies nicht der Fall, dann werden in Schritt (2) die Attribute stateValue, modified und stateCalculated gesetzt. Das Attribute stateValue, das den eigentlichen Status angibt, wird auf „nicht erfüllt“ (notFullfilled) gesetzt. Die Attribute modified und stateCalculated stellen Hilfsvariablen dar und werden auf false gesetzt.

In Schritt (3) wird geprüft, ob mindestens ein Systemelement der Funktion zugewiesen wurde. Ist dies der Fall, dann wird in Schritt (4) der Status der Funktion auf „teilweise realisiert“ (partlyFullfilled) gesetzt. Im Anschluss daran, oder falls noch keine Systemelemente der Funktion zugewiesen wurden, werden in Schritt (5) der Reihe nach alle untergeordneten Teilfunktionen ermittelt und solange das Attribut stateCalculated nicht auf true gesetzt ist, in den Schritten (6) bis (10) geprüft.

Gibt es keine untergeordneten Teilfunktionen, dann wird direkt mit Schritt (11) fortgefahren. In diesem Fall hätte die Funktion aufgrund der Schritte (2) bzw. (4) bereits den korrekten Status gemäß Tabelle 1 Nr. 2 und 3 aus Abschnitt 8.1.4.

In Schritt (6) wird das Statusobjekt der untergeordneten Teilfunktion ermittelt.

In Schritt (7) wird zuerst geprüft, ob der Status der untergeordneten Teilfunktion auf „teilweise realisiert“ (partlyFullfilled) gesetzt ist. Sollte das der Fall sein, dann wird der Status der eigenen Funktion ebenfalls auf „teilweise realisiert“ gesetzt und die Überprüfung kann beendet werden (stateCalculated := true). Dies entspricht der Regel 2.1 in Tabelle 2 in Abschnitt 8.1.4.

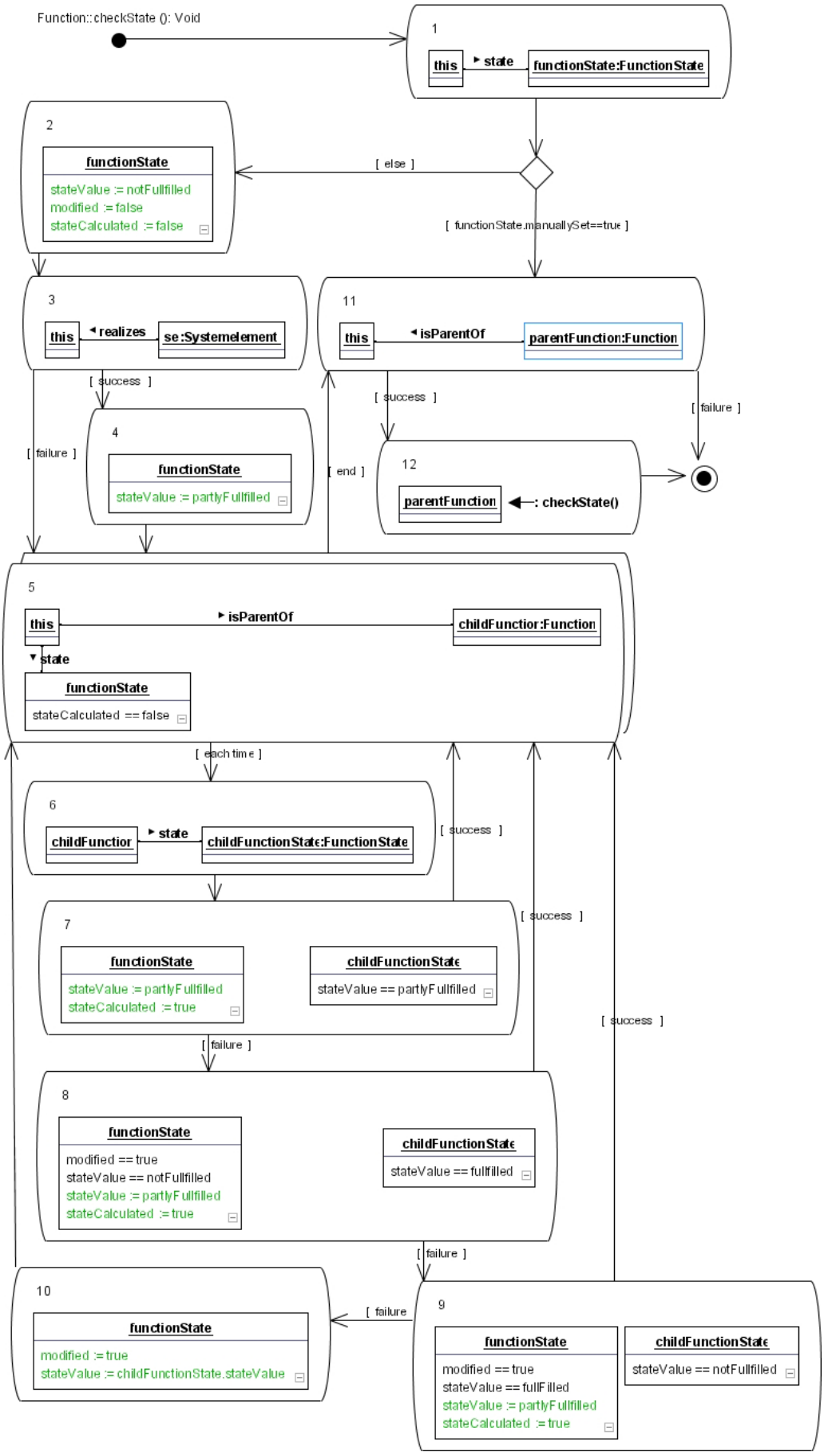


Abbildung 111: automatische Statusvergabe

In Schritt (8) wird geprüft, ob der Status der untergeordneten Teilfunktion auf „realisiert“ (fullfilled) gesetzt ist und der eigene Status auf „nicht realisiert“ (notFullfilled) und das Attribut modified auf true gesetzt ist. Das Attribut modified wird genau dann auf true gesetzt, sobald der Status innerhalb dieser Überprüfung erstmalig geändert wird. Diese Information ist für die Regel 2.2 aus Tabelle 2 in Abschnitt 8.1.4 erforderlich. Wurde also modified bereits auf true gesetzt und der eigene Status steht auf „nicht realisiert“, aber der Status der untergeordneten Teilfunktion steht auf „realisiert“, dann trifft die Regel 2.2 zu, der eigene Status wird auf „teilweise realisiert“ gesetzt und die Überprüfung kann beendet werden (stateCalculated := true).

Der Schritt (9) gleicht sehr stark dem Schritt (8). Hier sind lediglich der eigene Status und der Status der untergeordneten Teilfunktion vertauscht.

Zu Schritt (10) gelangt man bei der Überprüfung der ersten untergeordneten Teilfunktion, oder solange die Stati der untergeordneten Teilfunktionen entweder alle auf „realisiert“ oder „nicht realisiert“ stehen. In diesem Schritt wird zunächst das Attribut modified der eigenen Funktion auf true gesetzt und der eigene Status wird auf den Status der gerade betrachteten, untergeordneten Teilfunktion gesetzt (stateValue := childFunctionState.stateValue). Sollten die Schritt (7), (8) oder (9) nie zum erfolg führen (success), dann bedeutet das, dass alle untergeordneten Teilfunktionen den Status „realisiert“ oder „nicht realisiert“ besitzen. Da diesen Status dann auch die betrachtete Funktion besitzt, wurden die Regeln 1 bzw. 3 aus Tabelle 2 in Abschnitt 8.1.4 berücksichtigt.

Nachdem alle Teilfunktionen überprüft wurde, wird in Schritt (11) die übergeordnete Funktion („Vaterknoten“) ermittelt. Sollte es eine solche Funktion geben, d.h. die aktuelle Funktion ist nicht die Gesamtfunktion, dann wird für diese Funktion ebenfalls der Status mit Hilfe der Methode checkState überprüft und gegebenenfalls neu gesetzt (Schritt (12)).

8.13.3 Berücksichtigung der Regeln zur Vergabe der Stati

Im letzten Abschnitt wurde gezeigt, wie mit Hilfe der automatischen Statusvergabe die Regeln zur Vergabe der Stati aus den Tabellen 1 und 2 aus Abschnitt 8.1.4 angewandt werden. Jedoch bleibt die Regel 1 der Tabelle 1 dabei unbeachtet. Diese Regel besagt, dass eine Funktion ohne Teilfunktionen den Status „realisiert“ bekommt, wenn:

„Der Funktion mindestens ein Systemelement zugewiesen ist und dieses Systemelement automatisch aus der Bibliothek ermittelt wurde.“

Das das Systemelement, welches der Funktion zugewiesen wurde, automatisch aus der Bibliothek ermittelt wurde (vgl. Kapitel 7), ist der automatischen Statusvergabe nicht bekannt. Daher ist diese nicht in der Lage den entsprechenden Status zu vergeben. Um den Status dennoch entsprechend setzen zu können ist es erforderlich, dass nach der automatischen Suche und der anschließenden Auswahl eines Systemelements durch den Entwickler, die manuelle Statusvergabe für diese Funktion aktiviert wird. Dabei muss die Methode setStateManually der Klasse Function mit dem Übergabeparameter (FunctionState.fullfilled) aufgerufen werden. Auf diese Weise wird der Status der Funktion auf „realisiert“ gesetzt und die Regel 1 der Tabelle 1 wird berücksichtigt.

8.14 Beurteilung der Konsistenzsicherung

In den letzten Abschnitten wurden die verschiedenen Methoden inkl. der darin enthaltenen Graphtransformationsregeln beschrieben, mit deren Hilfe Modifikationen der Funktionshierarchie und Systemstruktur vorgenommen werden und gleichzeitig die geforderten Konsistenzbedingungen sicher gestellt werden.

Bei einigen Methoden ist es allerdings erforderlich, dass bei ihrer Ausführung einige Konsistenzregeln verletzt werden, damit andere Konsistenzregeln sicher gestellt werden können. In diesen Fällen müssen anschließend andere Methoden ausgeführt werden, um die verletzten Konsistenzregeln wieder zu korrigieren.

Im folgenden wird nun geprüft, ob durch die Ausführung der Methoden Konsistenzbedingungen verletzt werden, die anschließend nicht wieder korrigiert werden. Es wird geprüft, ob die Methoden in jedem Fall eine Funktionshierarchie und eine Systemstruktur erzeugen, die Konsistent zueinander sind.

Die Überprüfung erfolgt auf Basis der Aufrufreihenfolgen der einzelnen Methoden. Hierzu wird zunächst festgestellt, welche Methoden welche Konsistenzbedingungen verletzen und welche sie sicher stellen. Im Anschluss wird dann gezeigt, dass es keine Aufrufreihenfolge gibt, bei der eine Konsistenzbedingung verletzt wird, ohne sie anschließend zu korrigieren.

Abbildung 112 zeigt, welche Methoden welche Konsistenzregeln verletzen und welche Methoden welche Konsistenzregeln sicher stellen.

Legende

✓ stellt Konsistenzbedingung sicher

⊗ verletzt Konsistenzbedingung

	Konsistenzbedingung 1	Konsistenzbedingung 2	Konsistenzbedingung 3	Konsistenzbedingung 4
addSystemelement	⊗			
addRootFunction	⊗			
deleteRootFunction				
assignSystemelement	✓	⊗		
addHelpfunctions		✓		⊗
deleteChildFunction				
deleteSystemelement			⊗	
unassignSystemelement	⊗			⊗
deleteAllHelpfunctions			✓	
moveAssignment	✓			⊗
checkState				✓
moveHelpfunctions				
setStateManually				✓ ⊗
addChildFunction				✓
moveFunction				⊗

Abbildung 112: Konsistenzsicherung mit Hilfe von Graphtransformationsregeln

Jede Methoden ist so aufgebaut, dass sie bei der Verletzung einer Konsistenzregel automatisch diejenige Methode aufruft, die die entstandene Inkonsistenz wieder korrigiert. Hierbei ist zu beachten, dass die Ausführung einer Methode nicht immer auch zu einer

Verletzung einer Konsistenzbedingung führt. Bei einigen Methoden müssen erst bestimmte Bedingungen erfüllt sein, damit die Verletzung der Konsistenzregeln eintritt. So verletzt z.B. die Methode assignSystemelement die Konsistenzbedingung 2 nur dann, wenn das entsprechende Systemelement überhaupt Hilfsfunktionen besitzt.

Abbildung 113 zeigt die Reihenfolgen in der die Methoden im Einzelnen ausgeführt werden. Abgesehen vom Namen der Methode ist außerdem angegeben, welche Konsistenzbedingungen die Methode sicher stellt und welche sie verletzt bzw. verletzen könnte. Die einzelnen Aufrufe (Übergänge) werden in Form von Pfeilen dargestellt. Stehen an den Übergängen Bedingungen, dann werden die entsprechenden Methoden nur ausgeführt, wenn die Bedingung zutrifft. Sind keine Bedingungen an den Übergängen angegeben, dann wird die entsprechende Methode in jedem Fall ausgeführt.

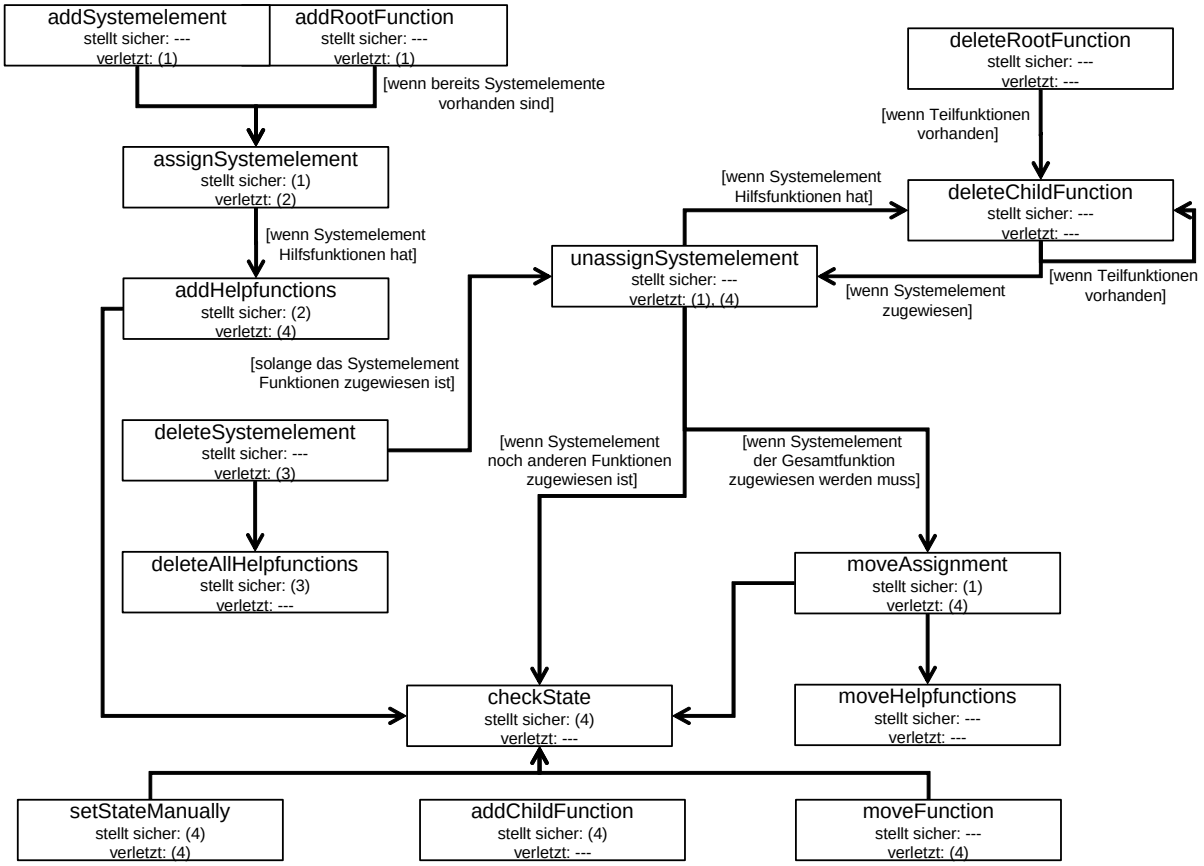


Abbildung 113: Mögliche Aufrufreihenfolgen der spezifizierten Methoden

Mit Hilfe der gezeigten Aufrufreihenfolgen wird anschaulich gezeigt, dass durch die Ausführung der Methoden die Konsistenzbedingungen eingehalten werden. Hierbei handelt es sich allerdings nicht um einen formalen Beweis, sondern ergibt sich aufgrund des Kontrollflusses innerhalb der aufgestellten Methoden.

Innerhalb der Methoden können sich Fehler eingeschlichen haben, die bisher nicht aufgefallen sind. Um auch dies hundertprozentig ausschließen zu können, müssten die Methoden bzw. die darin spezifizierten Graphtransformationsregeln mit Hilfe formaler Methoden, wie z.B. dem Model Checking, verifiziert werden.

8.15 Zusammenfassung

In diesem Kapitel wird gezeigt, wie die Funktionen der Funktionshierarchie und die Systemelemente der Systemstruktur zueinander in Beziehung stehen. Eine Funktion wird durch Zuweisung eines oder mehrerer Systemelemente realisiert. Werden alle Funktionen

durch Systemelemente realisiert, dann ist der Entwurf in Sinne der Funktionserfüllung vollständig spezifiziert.

Wesentlicher Inhalt dieses Kapitels stellt die Sicherstellung der aufgestellten Konsistenzbedingungen mit Hilfe von Graphtransformationsregeln dar. Das Ändern der Funktionshierarchie oder der Systemstruktur kann nur unter Berücksichtigung bestimmter Konsistenzbedingungen erfolgen, da ansonsten der Überblick darüber verloren geht, welche Funktionen von welchen Systemelementen bereits erfüllt werden. Es wird gezeigt, dass mit Hilfe der aufgestellten Graphtransformationsregeln sowohl die Funktionshierarchie, als auch die Systemstruktur modifiziert werden können und gleichzeitig die Konsistenzbedingungen eingehalten werden

KAPITEL 9: WERKZEUGUNTERSTÜTZUNG

Zur Unterstützung und Überprüfung des Ansatzes, ist im Rahmen dieser Arbeit ein entsprechendes Softwarewerkzeug entstanden. Dieses Werkzeug wurde in Form zweier Plug-Ins, dem Funktions-Hierarchie-Editor und dem Systemstruktur-Editor, auf Basis der Entwicklungsumgebung Eclipse entwickelt.

Im Folgenden wird zunächst die Entwicklungsumgebung Eclipse vorgestellt, gefolgt von den entwickelten Plug-Ins.

9.1 Entwicklungsumgebung Eclipse

Bei dem OpenSource Produkt Eclipse handelt es sich um eine frei konfigurierbare Entwicklungsumgebung (IDE[32]) zur Erstellung eigener Anwendungen. Eclipse stellt hierzu eine einheitliche graphische Oberfläche zur Verfügung, die mit Hilfe von Plug-Ins entsprechend erweitert bzw. konfiguriert werden kann. Mit Hilfe dieses Plug-In Konzepts können anwendungsspezifische Editoren implementiert werden, mit deren Hilfe dann ein individuelles Softwarewerkzeug entsteht.

Zur Entwicklung dieser Plug-Ins stellt Eclipse eine ganze Reihe von Grundfunktionalitäten bereit, die nicht mehr selbst entwickelt werden müssen. Hierzu zählen beispielsweise die standardisierte Oberfläche, einheitliche Zugriffsmechanismen auf Dateien, ein spezielles Sichtenkonzept, eine umfangreiche Grafikbibliothek uvm.

Das Eclipse Projekt wurde Ende 2001 von den Firmen IBM, Borland, Rational Software u.a. ins Leben gerufen. Derzeit gehören dem Konsortium über 30 namhafte Firmen an. Im Jahre 2004 wurde das Eclipse Projekt als ein „not-for-profit“ Projekt deklariert.

Abbildung 114 zeigt beispielhaft die Eclipse-Oberfläche. Zu sehen ist ein Plug-In zur Entwicklung von Java-Anwendungen. Auf der linken und rechten Seite befinden sich zwei unterschiedliche Sichten, die verschiedene Aspekte bei der Entwicklung von Java-Anwendungen zeigen. Auf der linken Seite sind die einzelten Source-Code Dateien aufgelistet und auf der rechten Seite befindet sich der Inhalt dieser Dateien.

Die Anordnung und das Aussehen der Sichten und der Menüs wird von Eclipse gesteuert. Das bringt einen erheblichen Vorteil bei der Entwicklung, da man sich darum nicht explizit kümmern muss.

[32] IDE: Integrated Development Environment

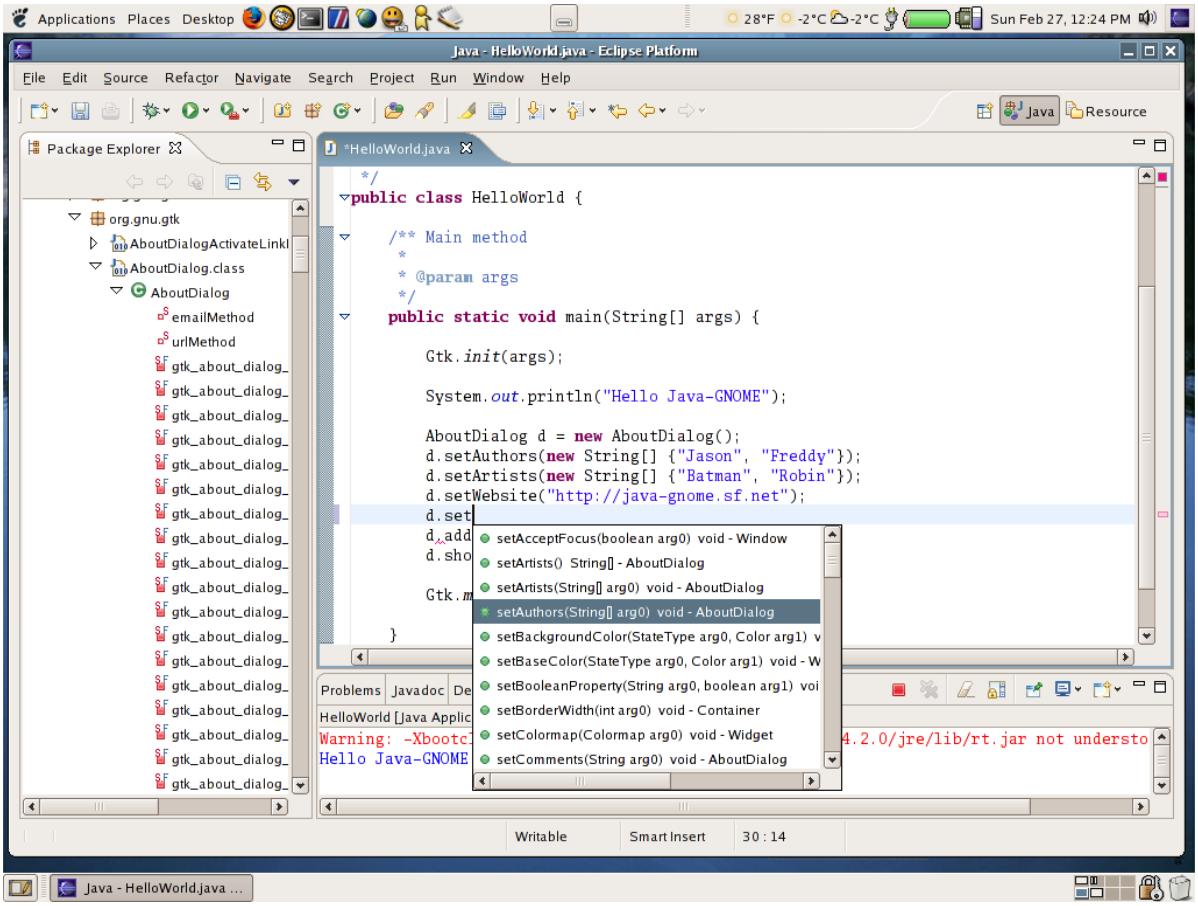


Abbildung 114: Entwicklungsumgebung Eclipse

Im Folgenden werden nun die beiden Eclipse Plug-Ins vorgestellt, die im Rahmen dieser Arbeit entwickelt wurden.

9.2 Funktions-Hierarchie-Editor

Das erste entwickelte Plug-In ist der Funktions-Hierarchie-Editor (FH-Editor). Mit Hilfe dieses Plug-Ins ist es möglich Funktionshierarchien zu modellieren, wie sie in Kapitel 5 vorgestellt wurden.

9.2.1 Oberfläche

Der FH-Editor besteht aus drei Bereichen, der Symbolleiste, dem Zeichenbereich und dem Property-Bereich(vgl. Abbildung 115):

Symbolleiste: Mit Hilfe von Schaltflächen der Symbolleiste können Funktionen angelegt bzw. gelöscht und die hierarchischen Beziehungen festgelegt werden.

Zeichenbereich: Im Zeichenbereich wird die Funktionshierarchie grafisch dargestellt.

Property-Bereich: Im Property-Bereich werden den Eigenschaften der Funktionen konkrete Werte zugewiesen. Hierzu zählen z.B. die Substantive, das Verb oder der Name der Funktion. Zusätzlich kann auch der Realisierungsgrad (nicht realisiert, teilweise realisiert und realisiert) festgelegt werden. Dieser wird dann in Form unterschiedlicher Farben (rot, gelb, grün) im Zeichenbereich dargestellt.

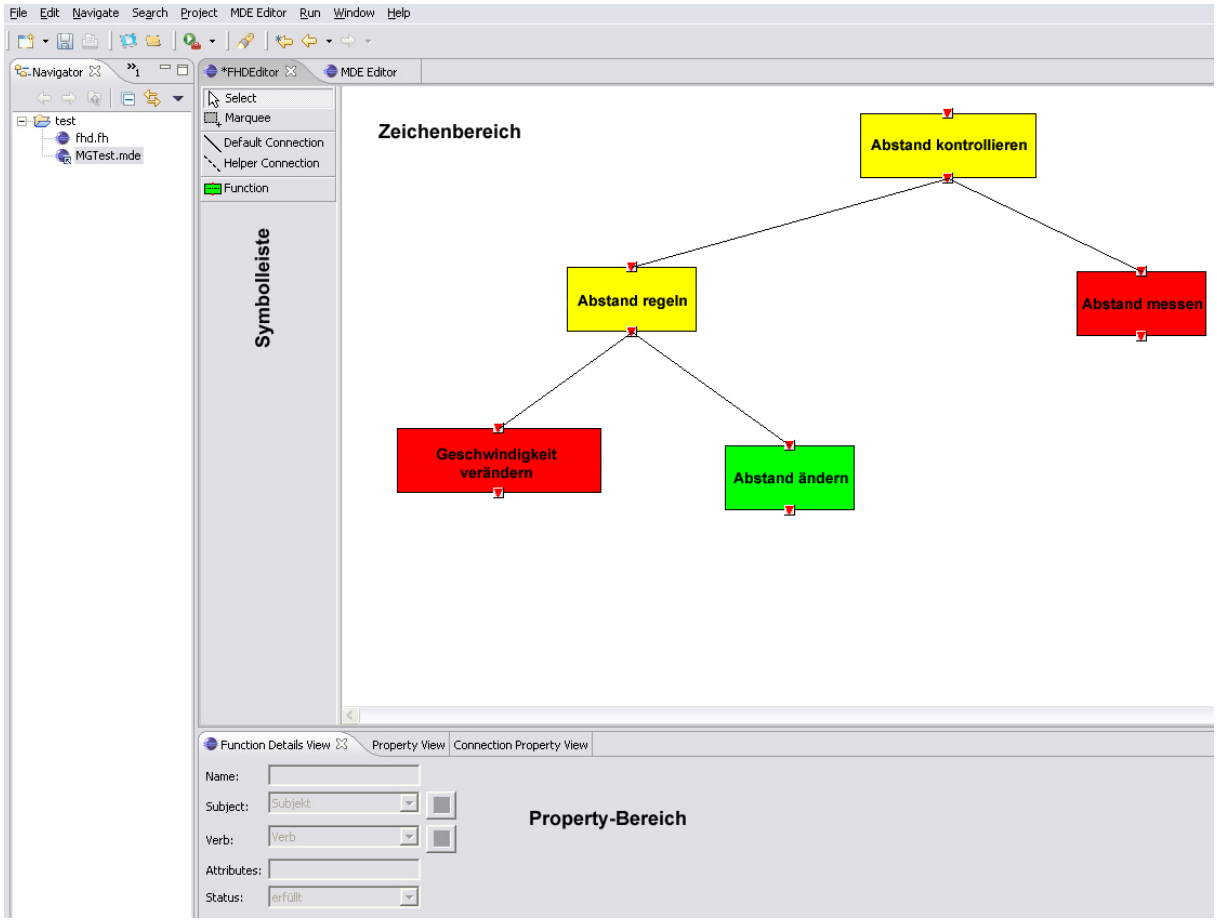


Abbildung 115: Funktions-Hierarchie Editor (FH-Editor)

9.2.2 Features

Der FH-Editor wurde mit einigen Features versehen, die den Entwickler bei der Modellierung der Funktionshierarchie unterstützen.

Bei der Zuweisung von Substantiven und Verben unterstützt der FH-Editor, indem er entsprechende Listen anzeigt, aus denen der Entwickler dann die benötigten Substantive und Verben auswählen kann.

Der FH-Editor stellt sicher, dass nur reine Baumstrukturen (Hierarchien) erzeugt werden können. Es ist beispielsweise nicht möglich zwei Funktionen in einer untergeordneten Teilfunktion, wieder zusammenzuführen.

Der FH-Editor überprüft, ob alle Funktionen miteinander verbunden sind. Ist dies nicht der Fall, wird eine entsprechende Hinweismeldung angezeigt.

Der FH-Editor ist in der Lage den Realisierungsgrad der Funktionen automatisch zu berechnen (vgl. Kapitel 8.13.2).

9.2.3 Schnittstelle

Um Informationen zwischen den beiden Plug-Ins auszutauschen und um insbesondere die in Kapitel 8 beschriebene Konsistenz sicher zu stellen, wurde eine entsprechende Schnittstelle entwickelt. Über diese Schnittstelle werden dem FH-Editor prinzipiell zwei unterschiedliche Arten von Informationen mitgeteilt.

Zum einen wird dem FH-Editor mitgeteilt, welchen Funktionen welche Systemelemente zugewiesen wurden. Dies ist u.a. bei der Bestimmung des Realisierungsgrades wichtig.

Zum anderen wird dem FH-Editor mitgeteilt, welche Modifikationen an der Funktionshierarchie durchgeführt werden sollen. Wird zum Beispiel ein Systemelement in die Systemstruktur (vgl. Abschnitt 9.3) integriert und dieses Systemelement Hilfsfunktionen benötigt, dann müssen diese Hilfsfunktionen in der Funktionshierarchie angezeigt werden.

Über die entwickelte Schnittstelle ist der FH-Editor nicht nur in der Lage Informationen zu empfangen, er kann auch Informationen senden. Welche Informationen dies sind und welche Auswirkungen sie auf den Sys-Editor haben, wird in Abschnitt 9.3 erläutert.

Zusätzlich wird über die Schnittstelle dem FH-Editor mitgeteilt, welche Funktion graphisch hervorgehoben werden soll. Dies ist dann von Interesse, wenn im Sys-Editor ein Systemelement markiert wurde und man wissen möchte, welchen Funktionen dieses Systemelement zugewiesen wurde.

9.3 Systemstruktur-Editor

Das zweite entwickelte Plug-In ist der Systemstruktur-Editor (Sys-Editor). Mit Hilfe dieses Plug-Ins ist es möglich die Systemstruktur eines mechatronischen Systems, gemäß Kapitel 6, zu modellieren.

9.3.1 Oberfläche

Die Oberfläche des Sys-Editor besteht aus insgesamt vier Bereichen, der Systemelementbibliothek, der Symbolleiste, dem Zeichenbereich und dem Property-Bereich (vgl. Abbildung 116).

Systemelementbibliothek: In diesem Bereich werden die, in der Bibliothek hinterlegten Systemelemente, angezeigt. Aus diesen Elementen kann der Entwickler die benötigten Elemente auswählen und per Drag-and-Drop in den Zeichenbereich integrieren.

Jedes Systemelement wird in einem eigenen XML-Dokument gespeichert. In diesem XML-Dokument sind alle erforderlichen Daten bzgl. des Systemelements hinterlegt. Hierbei handelt es sich sowohl um den strukturellen Aufbau, als auch um Information, welche Funktionen es prinzipiell erfüllen kann und welche Hilfsfunktionen es benötigt.

Symbolleiste: Mit Hilfe der Schaltflächen der Symbolleiste lassen sich Systemelemente erzeugen bzw. löschen. Hier lassen sich auch neue Verbindungen zwischen zwei Systemelementen erzeugen bzw. wieder löschen.

Zeichenbereich: In diesem Bereich wird die Systemstruktur angezeigt. Jedes Systemelement wird als Sechseck dargestellt inkl. seiner Ports und entsprechenden Verbindungen. Mit Hilfe eines Kontext-Menüs lassen sich einige Einstellungen an den Systemelementen vornehmen.

Property-Bereich: In diesem Bereich werden die verschiedenen Eigenschaften der Systemelemente und der Verbindungen angezeigt und modifiziert.

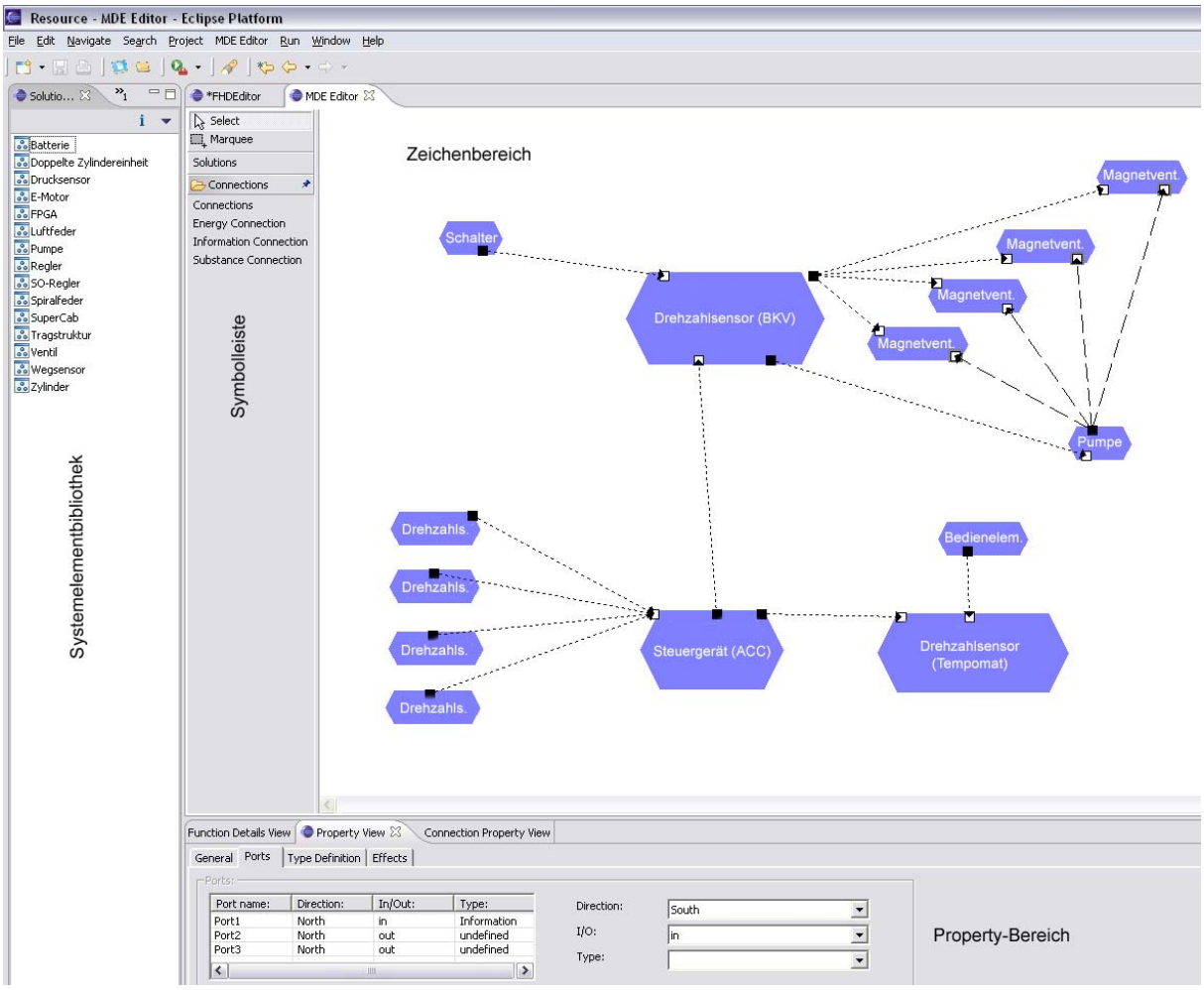


Abbildung 116: Systemstruktur-Editor (SyS-Editor)

9.3.2 Features

Wie auch der FH-Editor ist der Sys-Editor mit einigen Features ausgestattet, die den Entwickler bei der Modellierung der Systemstruktur unterstützen.

Der Sys-Editor überprüft, dass nur Ports vom gleichen Typ miteinander verbunden werden. Sollen zwei inkompatible Ports verbunden werden, so lässt der Sys-Editor dies nicht zu.

Der Sys-Editor überprüft, ob alle Ports verbunden sind. Ist dies nicht der Fall, wird eine entsprechende Hinweismeldung ausgegeben.

Der Sys-Editor überprüft, ob jedes Systemelement mindestens einer Funktion im FH-Editor zugewiesen wurde. Ist dies nicht der Fall, wird eine entsprechende Hinweismeldung ausgegeben.

Der Sys-Editor ist in der Lage, auf Basis einer gegebenen Funktion, passende Systemelemente zu ermitteln (vgl. Kapitel 7). Bei der Ermittlung dieser Systemelemente werden die entsprechenden XML-Dokumente durchsucht und eine Liste passender Systemelemente dem Benutzer angezeigt. Sobald der Benutzer ein Systemelement ausgewählt hat, wird das Element in die Systemstruktur integriert. Gleichzeitig wird dem FH-Editor mitgeteilt zu welcher Funktion das ausgewählte Systemelement gehört. Der FH-Editor erweitert dann die Funktionshierarchie, falls dies aufgrund benötigter Hilfsfunktionen nötig ist.

9.3.3 Schnittstelle

Wie bereits in Abschnitt 9.2.3 erläutert, wurde jeder Editor mit einer Schnittstelle ausgestattet über die er mit dem jeweils anderen Editor kommunizieren kann.

Über die Schnittstelle wird dem Sys-Editor mitgeteilt, für welche Funktion ein Systemelement aus der Bibliothek ermittelt werden soll (vgl. 9.3.2).

Über die Schnittstelle wird dem Sys-Editor mitgeteilt, welche Systemelemente optisch hervorgehoben werden sollen. Dies ist dann hilfsreich, wenn man im FH-Editor eine Funktion markiert und wissen möchte, welche Systemelemente dieser Funktion zugewiesen sind.

9.4 Gemeinsamer Einsatz der Editoren

Abgesehen von der Tatsache, dass beide Editoren unabhängig voneinander benutzt werden können, ist deren gemeinsame Verwendung jedoch notwendig, um die Konsistenz der Modelle sicherstellen zu können.

Um einen besseren Überblick zu bekommen, bietet Eclipse die Möglichkeit die Sichten der einzelnen Editoren gleichzeitig anzuzeigen. Abbildung 117 zeigt die beiden Editoren innerhalb der Eclipse-Umgebung. Im oberen Teil befindet sich der FH-Editor und im unteren Teil der Sys-Editor. Auf diese Weise bekommt man direkt angezeigt, welche Auswirkungen eine Modelländerung innerhalb des einen Editors, auf das Modell des anderen Editors, hat.

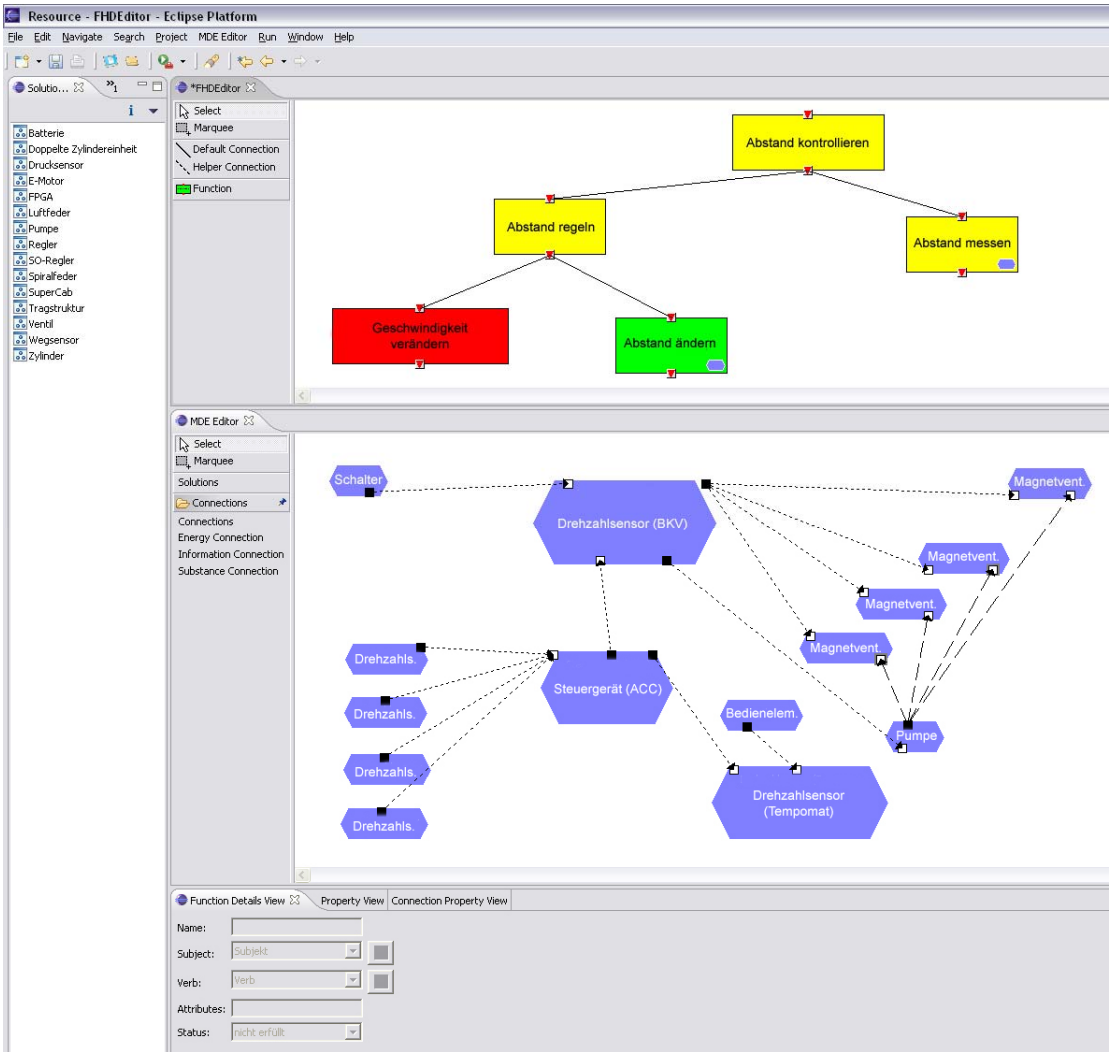


Abbildung 117: Werkzeugunterstützung bei der Suche nach Systemelementen

Sowohl die in Kapitel 7 beschriebene Suche nach Systemelementen, als auch die Sicherstellung der Konsistenz (vgl. Kapitel 8) ist nur möglich, wenn beide Editoren gleichzeitig eingesetzt werden.

Schlussbemerkung: Abschließend ist zu erwähnen, dass noch nicht alle Konzepte dieser Arbeit in den beiden Plug-Ins umgesetzt werden konnten. Hier dauert die Entwicklung derzeit noch an.

KAPITEL 10: ZUSAMMENFASSUNG UND AUSBLICK

Der Entwurf mechatronischer Systeme wird mit zunehmender Verknüpfung der beteiligten Disziplinen immer schwieriger. Ausgehend von einer Aufgabenstellung müssen Entwicklerteams das mechatronische System entwerfen, wobei sie alle Vor- und Nachteile der unterschiedlichen Disziplinen berücksichtigen müssen. Problem dabei ist, dass sie sich sehr stark auf ihr Wissen und ihre Erfahrung verlassen müssen, da es meistens noch keine adäquate Methodik inkl. einer entsprechenden Rechnerunterstützung gibt.

Eine der daraus hervorgehenden Problemstellungen ist die Modellierung der funktionalen Anforderungen mit Hilfe einer Funktionshierarchie und der systematische Übergang zur Systemstruktur. Die Funktionshierarchie ermöglicht das systematische Zerlegen der funktionalen Anforderungen in überschaubare Funktionen, denen Systemelemente zu ihrer Realisierung zugewiesen werden können.

Innerhalb der Systemstruktur sind diejenigen Systemelemente aufgeführt, die in der Lage sind, die Funktionen zu realisieren. Heutzutage erfolgt die Auswahl dieser Systemelemente jedoch ausschließlich durch den Entwickler bzw. das Entwicklerteam. Dabei entstehen drei prinzipielle Probleme.

(1) Es werden nur solche Systemelemente verwendet, die den Entwicklern bekannt sind. Möglicherweise bessere, aber den Entwicklern nicht bekannte Systemelement, werden nicht in Betracht gezogen.

(2) Es werden nur diejenigen Systemelemente betrachtet die die gestellten, funktionalen Anforderungen unmittelbar erfüllen. Die durch die Wahl der Systemelemente entstehenden Abhängigkeiten bleiben unberücksichtigt. Dies kann während der eigentlichen Entwicklung, aufgrund aufwändiger Anpassungen, zu erheblichen Kosten führen.

(3) Aufgrund ständiger Änderungen, sowohl der funktionalen Anforderungen als auch der Systemstruktur, geht mit zunehmender Komplexität der Überblick verloren. Es kann nicht mehr nachvollzogen werden, ob alle Funktionen erfüllt werden, oder ob Systemelemente genutzt werden sollen, die gar keine Funktionen mehr erfüllen.

Um diese geschilderten Probleme zu lösen, wurde im Rahmen dieser Arbeit der Entwurfsprozess für mechatronische Systeme analysiert. Auf Grundlage der VDI-Richtlinien 2221 [VDI93], 2222 [VDI97], 2206 [VDI04] und Arbeiten von Pahl/Beitz [PB04], Ehrlenspiel [Ehr03], Roth [Rot00] und Kallmeyer [Kal98] wurden die Funktionsmodellierung und die Systemstrukturmodellierung als entscheidende Ansatzpunkte identifiziert. Insbesondere die Arbeit von Kallmeyer dient dabei als Grundlage.

Im Rahmen der Funktionsmodellierung werden die funktionalen Anforderungen in Form einer Hierarchie abgebildet. Ausgehend von der Gesamtfunktion wird diese Schritt für Schritt in Teilfunktionen zerlegt, bis für die Teilfunktionen ein oder mehrere Systemelemente ermittelt werden können, die diese Funktion erfüllen.

Bei den Systemelementen handelt es sich entweder um maschinenbauliche Erzeugnisse wie ein Zylinder oder ein Motor, um elektrotechnische Erzeugnisse wie ein Steuergerät oder Rechner, oder um informationstechnische Erzeugnisse, wie z.B. ein Regler. Jedes dieser Systemelemente ist alleine, oder in Kombination mit anderen in der Lage die funktionale

Anforderungen zu erfüllen. Mit Hilfe entsprechender Schnittstellen werden die einzelnen Systemelemente zu einer Systemstruktur verknüpft.

10.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde der Zusammenhang zwischen der Funktionshierarchie und der Systemstruktur auf Basis von Graphtransformationsregeln formal spezifiziert. Grundlage hierfür war die vorhergehende Formalisierung der Funktionshierarchie und der Systemstruktur. Beide lagen bis zu Beginn dieser Arbeit nur informal vor.

Bei der Formalisierung der Funktionshierarchie wurde darauf geachtet, dass nicht zu viele Einschränkungen gemacht wurden. Der Entwickler sollte nach wie vor, bei der Beschreibung der Funktionen, den ihm bekannten Wortschatz nutzen können.

Jede Funktion wird mit Hilfe von Substantiven und einem Verb beschrieben. Die Substantive beschreiben dabei sowohl den Input als auch den Output einer Funktion. Das Verb beschreibt die Art der Transformation zwischen dem Input und dem Output.

Sowohl die Substantive, als auch die Verben können beliebig gewählt werden. Hierbei ist lediglich im Vorfeld zu definieren, ob bestimmte Substantive bzw. Verben untereinander in Beziehung stehen. Es ist anzugeben, ob beispielsweise zwei Verben die gleiche Bedeutung haben (z.B.: „senden“ und „übertragen“), oder ob das eine Verb eine konkretere Form eines anderen Verbs darstellt (z.B.: „ändern“ und „vergrößern“).

Die statische Struktur der Funktionshierarchie wurde mit Hilfe von UML-Klassendiagrammen formal beschrieben.

Mit der Arbeit von Kallmeyer wurde erstmalig eine Notation zur Beschreibung der Systemstruktur eingeführt. Die Beschreibung der Notation war allerdings nur informal und über weite Strecken unvollständig oder ungenau. In dieser Arbeit wurde daher zunächst die Systemstruktur konkretisiert. Zum einen wurden neue Konstrukte definiert, zum anderen wurde festgelegt, wie die Verknüpfung der Systemelemente untereinander spezifiziert werden dürfen. Diese Festlegungen wurden mit Hilfe von UML-Klassendiagrammen für die statische Struktur und mit Hilfe umgangssprachlich definierter Regeln für das Verhalten spezifiziert.

Darüber hinaus wurde eine Verknüpfung zur Funktionsmodellierung geschaffen, indem Systemelemente in die Lage versetzt wurden Funktionen zu realisieren. Dies wurde ebenfalls mit Hilfe von UML-Klassendiagrammen beschrieben.

Um das oben beschriebene Problem Nr. 1 zu lösen wurde ein Suchalgorithmus entwickelt, der in der Lage ist, für eine gegebene Funktion alle Systemelemente zu ermitteln, die diese Funktion realisieren können. Grundlage hierfür ist eine entsprechend große Menge von Systemelementen auf die der Algorithmus zugreifen kann. Daher wurde festgelegt, dass eine Systemelementbibliothek zur Verfügung stehen muss, in der Systemelemente abgelegt werden können.

Zur Beantwortung des zweiten Problems wurden Konsistenzbedingungen zwischen der Funktionshierarchie und der Systemstruktur definiert und mit Hilfe von UML-Klassendiagrammen und Graphtransformationsregeln formal beschrieben. Alle Änderungsmöglichkeiten der Funktionshierarchie und der Systemstruktur wurde genau festgelegt, damit im Laufe des Entwurfs der Überblick nicht verloren geht.

Darüber hinaus wurde spezifiziert, dass jedes Systemelement, welches in der Systemelementbibliothek hinterlegt ist, inkl. seiner funktionalen Abhängigkeiten abgelegt wird. Dies hat den Vorteil, dass diese Abhängigkeiten nach Auswahl des Systemelements direkt in der Funktionshierarchie angezeigt werden können. Dadurch können keine

Abhängigkeiten vergessen werden, deren Erfüllung dann im weiteren Verlauf der Entwicklung erhebliche Kosten verursachen.

Das dritte Problem wird dadurch gelöst, dass jeder Funktion ein Status zugewiesen wird. Dieser Status gibt an, ob die Funktion bereits von Systemelementen realisiert, teilweise realisiert, oder noch gar nicht realisiert wird.

Die Vergabe dieses Status erfolgt mit Hilfe eines entsprechenden Algorithmus weitestgehend automatisch. Hierzu wurden Regeln aufgestellt, die angeben, welcher Status jeweils zu vergeben ist.

10.2 Ausblick

Mit der Formalisierung der Konsistenz zwischen der Funktionshierarchie und der Systemstruktur konnte eine bis dahin bestehende Lücke im Entwurf mechatronischer Systeme geschlossen werden. Allerdings bleiben auch hier noch eine Reihe weiterer Punkte offen, die das Vorgehen weiter verbessern könnten.

Ein erster Punkt ist die Erweiterung der Formalisierung der Systemstruktur um Plausibilitätsprüfungen. Dadurch könnten fehlerhafte Verknüpfungen von Systemelementen entdeckt bzw. verhindert werden. Dies wäre mit Hilfe von Graphtransformationsregeln möglich, da bereits ein UML-Klassendiagramm als Grundlage zur Verfügung steht.

Ein weiterer Punkt der bearbeitet werden sollte, ist die Unterstützung des Entwicklers beim Befüllen der Systemelementbibliothek. Hier wäre es denkbar, eine Analyse der hinzuzufügenden Systemelemente vorzunehmen und zu ermitteln, welche Funktionen sie möglicherweise erfüllen können. Dies könnte auf Basis der Input- und Output-Ports der Systemelemente und auf Basis entsprechender Regeln erfolgen. Eine Regel könnte beispielsweise lauten, dass ein bestimmter Inputport und ein bestimmter Outputport vorhanden sein müssen, damit das Systemelement eine bestimmte Funktion erfüllen kann.

Eine andere Möglichkeit, um herauszufinden welche Funktionen ein Systemelement erfüllt besteht, wenn festgehalten wird, welche Systemelemente welchen Funktionen zugewiesen wurden. Wird ein Systemelement einer Funktion zugewiesen, so wird diese Zuweisung gespeichert und das System ist bei einer erneuten Suche in der Lage dieses Systemelement in die Suche mit einzubeziehen. Dadurch wäre ein System in der Lage selbst zu lernen, welche Funktion von welchen Systemelementen erfüllt werden können.

Durch die Formalisierung im Rahmen dieser Arbeit ist sichergestellt, dass es einen Übergang von der Funktionshierarchie zur Systemstruktur gibt. Die so entstehende Systemstruktur ist dann zwar in der Lage die Funktionen der Funktionshierarchie zu erfüllen, jedoch stellt sich die Frage, ob es nicht noch eine bessere Alternative gibt. Also ob eine Systemstruktur aufgestellt werden kann, die gemäß definierter Kriterien eine bessere, vielleicht sogar optimale Variante darstellt.

Eine Lösung hierfür könnte auf Grundlage von gegenseitigen Verträglichkeiten zwischen den Systemelementen definiert werden. Jedes Systemelement erhält dabei einen bestimmten Verträglichkeitswert gegenüber jedem anderen Systemelement. Auf Basis dieser Werte könnte nun ein Algorithmus eine optimale Systemstruktur ermitteln. Ein erster Ansatz hierzu wurde bereits auf der „Mechatronics and Robotics 2004“ [GST04] vorgestellt.

Eine andere Form der Optimierung ist denkbar, wenn die Suche nach Systemelementen in der Form eingeschränkt werden könnte, dass nur unten den bereits in der Systemstruktur vorhandenen Systemelementen gesucht wird. Auf diese Weise könnte sich die Anzahl der verwendeten Systemelemente möglicherweise reduzieren. Erfolgreich anwendbar ist dieser Ansatz jedoch erst ab einer entsprechend großen Systemstruktur.

Betrachtet man den gesamten Entwicklungsprozess stellt man fest, dass mit der Spezifizierung der Systemstruktur die prinzipielle Lösung des mechatronischen Systems feststeht. Als nächstes schließt sich dann der domänenspezifische Entwurf an, d.h. jede Disziplin extrahiert aus der Systemstruktur diejenigen Teile, die für ihre Arbeit erforderlich sind. Aus diesem Grund wäre es sinnvoll, einen automatisierbaren Übergang zwischen der Systemstruktur und den Modellen der sich anschließenden Disziplinen zu schaffen [Val04].

Einer dieser Übergänge könnte der Übergang zu den Arbeiten der Disziplin Regelungstechnik/Informationstechnik sein. So wäre es z.B. sinnvoll, gewisse Systemelemente als Vorgaben für den Reglerentwurf zu nutzen. Es könnten beispielsweise Systemelemente der Systemstruktur, die einen Regler repräsentieren inkl. ihrer Verknüpfungen, als Vorlage in die Systeme zur Modellierung von Reglern übertragen werden.

Erste Ideen in diese Richtung betrachten die Verknüpfung der Systemstruktur mit der Modellierung von hybriden Statecharts in der Entwicklungsumgebung Fujaba. Ausgewählte Systemelemente sollen hier die strukturelle Grundlage für die weitere Ausarbeitung diskret/kontinuierlicher Reglerstrukturen darstellen.

Neben dem Übergang zur Informationstechnik wäre auch der Übergang zur Elektrotechnik bzw. Mechanik sinnvoll. Erste Überlegungen in diese Richtung betrachten die von den Systemelementen erzeugten Effekte (z.B.: Wärme, Magnetismus etc.). Einige Systemelemente erzeugen diese Effekte, andere werden von ihnen beeinflusst. Hier wäre es möglich, die in der Systemstruktur modellierten Systemelemente um Effekte und Anfälligkeiten zu ergänzen und zu berechnen, in welchem Abstand die Systemelemente mindestens angeordnet werden müssten, damit die Effekte keine negativen Auswirkungen haben. Das Ergebnis dieser Berechnung könnte dann als Vorlage für die endgültige, dreidimensionale Ausarbeitung mit Hilfe von CAD-Systemen dienen.

ANHANG: LAUFZEITBETRACHTUNG

In diesem Abschnitt wird das Laufzeitverhalten des Suchalgorithmus aus Kapitel 7 betrachtet. Dies erfolgt in Form inhaltlicher Argumentation und nicht durch einen formalen Beweis. Betrachtet wird vor allem der worst-case. Dabei wird festgestellt, dass der Algorithmus ein sehr schlechtes Laufzeitverhalten aufweist. Dieses kann jedoch mit Hilfe konzeptioneller Überlegungen relativiert werden. Ausgangspunkt für die Überlegungen sind das UML-Klassendiagramm in Abbildung 63 und das Story-Diagramm aus Abbildung 68.

Entscheidenden Einfluss auf die Komplexität und damit die Laufzeit haben die Mengen der Verben (Klasse: VerbName), Substantive (Klasse: SubjectName), Systemelemente (Klasse: Systemelement) und die Funktionen die ein Systemelement erfüllt (Klasse: SE_Function).

Es wird angenommen, das im worst-case zu jeder Gruppe von äquivalenten Verben (Klasse: VerbGroup) und äquivalenten Substantiven (Klasse: SubjectGroup) genau ein Verb bzw. genau ein Substantiv gehört. Dies bedeutet das die Anzahl der Objekte gleich groß ist.

Folgende Mengen werden betrachtet:

VN = Menge aller Verben mit |VN| = Anzahl der Verben

VG = Menge aller Verbgruppen mit |VG| = Anzahl der Verbgruppen

SN = Menge aller Substantive mit |SN| = Anzahl der Substantive

SG = Menge aller Substantivgruppen mit |SG| = Anzahl der Substantivgruppen

SY = Menge aller Systemelemente mit |SY| = Anzahl der Systemelemente

SE = Menge aller Funktionen die durch Systemelemente erfüllt werden, mit |SE| = Anzahl dieser Funktionen

StoryPattern (1): Im ersten Story-Pattern des Algorithmus wird genau eine Anweisung ausgeführt. Diese ist unabhängig von der Anzahl der oben erwähnten Mengen und kann daher vernachlässigt werden.

StoryPattern (2): Im zweiten Story-Pattern werden ausgehend von einer Verbgruppe über die Assoziation isParentOf alle anderen Verbgruppen ermittelt. Dies erfolgt rekursiv, wobei jede Verbgruppe genau einmal „besucht“ wird, da im worst-case die aktuelle Verbgruppe eine Konkretisierung aller anderen Verbgruppen darstellt. Dadurch ergibt sich eine Komplexität von |VG|.

StoryPattern (3): Im dritten Story-Pattern wird zunächst für jede Verbgruppe ermittelt, welche Verben zu der Gruppe gehören. Dann wird für jedes Verb ermittelt, durch welche Systemelemente es erfüllt wird. Die Gesamtheit aller Funktionen die durch die Systemelemente erfüllt werden ist |SE|. Da im worst-case jede Verbgruppe genau ein Verb hat, ergibt sich eine Komplexität von (|VG| + |VN|) * |SE|.

StoryPattern (4): Im vierten Story-Pattern werden Hilfsobjekte für jedes Input- und Output-Substantiv angelegt. Für die Hilfsobjekte der Input-Substantive ergibt sich eine Komplexität von (|SG| + |SN|) * |SG|. Gleicher Aufwand entsteht für die Output-Substantive, wodurch sich eine Komplexität von 2 * (|SG| + |SN|) * |SG| ergibt.

StoryPattern (5): Im letzten Story-Pattern werden die geforderten In- und Output-Substantive mit den In- und Output-Substantiven der Systemelemente verglichen. Für den Vergleich der Input-Substantive ergibt sich im worst-case eine Komplexität von $|SN|^2 * |SG| *$

|SE|. Da dies auch für die Output-Substantive durchgeführt wird, ergibt sich insgesamt eine Komplexität des letzten Story-Pattern von 2 * (|SN|² * |SG| * |SE|).

Insgesamt ergibt sich somit für den gesamten Algorithmus eine Komplexität von:

2 * (|SN|² * |SG| * |SE|) + 2 * (|SG| + |SN|) * |SG| + (|VG| + |VN|) * |SE| + |VG|

Aufgrund der Tatsache das im worst-case die Anzahl der Verben gleich der Anzahl der Verbgruppen ist und die Anzahl der Substantive gleich der Anzahl der Substantivgruppen ist, setzen wir: |VG| = |VN| und |SG| = |SN|, wodurch sich die folgende Komplexität ergibt:

2 * (|SN|³ * |SE|) + 4|SN|² + |VN|² * |SE| + |VN|

Noch nicht genau betrachtet wurde die Menge |SE|, also die Menge der Funktionen die von den Systemelementen erfüllt werden können. Diese Menge stellt eine Kombination von Input-Substantiven, Verben und Output-Substantiven dar. Im worst-case ergibt sich eine Anzahl von $2^{|SN|}$ Input-Substantive und $2^{|SN|}$ Output-Substantive. Kombiniert mit der Anzahl der Verben ergibt sich eine Komplexität für die Menge |SE| von: $|SE| = 2^{|SN|} * |VN| * 2^{|SN|} = |VN| * 2^{2 * |SN|}$.. Ersetzt man nun |SE| aus obiger Formel so ergibt sich insgesamt folgende Komplexität:

Worst-Case: $2 * (|SN|^3 * |VN| * 2^{2 * |SN|}) + 4|SN|^2 + |VN|^3 * 2^{2 * |SN|} + |VN|$

Dies ist eine Abschätzung der Komplexität und nicht formal bewiesen. Sie zeigt jedoch deutlich, dass die Laufzeit des Algorithmus exponentiell von der Anzahl der Substantive abhängt.

In Kapitel 7 wurde gezeigt, dass das exponentielle Wachstum im praktischen Einsatz keinen signifikanten Einfluss auf die Laufzeit hat. Dies liegt insbesondere daran, dass das worst-case Szenario in der Praxis, aller Voraussicht nach, nicht eintreten wird.

Konzeptionelle Betrachtung der Laufzeit

Die Laufzeitbetrachtung beruht auf der Annahme, dass die gesuchte Funktion alle verfügbaren Substantive als Input und alle verfügbaren Substantive als Output besitzt. Ferner geht die Abschätzung davon aus, dass sowohl jedes Verb mit allen anderen Verben untereinander in Beziehung steht, als auch jedes Substantive mit allen anderen Substantiven in Beziehung steht. Des Weiteren wird angenommen, das jedes Verb eine eigene Verbgruppe und jedes Substantiv eine eigene Substantivgruppe besitzt. Schließlich wird angenommen, dass jedes Systemelement jede mögliche Kombination von Funktionen erfüllt.

Diese Annahmen stellen das absolute worts-case Szenario dar. Unter der Voraussetzung, dass es eine ausreichend große Menge an Substantive und Verben gibt, kommt dieses Szenario im praktischen Einsatz nicht vor.

Folgende Erkenntnisse sind daher entscheidend für die Beurteilung des Algorithmus:

(1) Es gibt keine vollständig zusammenhängenden Mengen von Verben und Substantiven. Es gibt immer nur kleinere Gruppen.

(2) Die Funktionen einer Funktionshierarchie haben eine wesentlich geringere Anzahl an Input- und Output-Substantive, als im worst-case Szenario angenommen.

(3) Die Systemelemente erfüllen in der Regel eine deutlich geringere Anzahl an unterschiedlichen Funktionen.

(4) Nicht jedes Verb bzw. Substantiv stellt eine eigene Gruppe dar. Dies würde bedeuten, dass es keine äquivalenten Verben bzw. Substantive gibt, was die Untersuchungen innerhalb dieser Arbeit jedoch ausschließen.

Aufgrund dieser Erkenntnisse kann man im durchschnittlichen Fall von einer Laufzeit ausgehen, die für die meisten Entwickler akzeptabel sein sollten.

LITERATURVERZEICHNIS

[Alt73] Altshuller, G.S.: Erfinden - (k)ein Problem? Anleitung für Neuerer und Erfinder, Berlin, Verlag Tribüne, 1973

[Alt84] Altshuller, G.S.: Creativity as an exact science: the theory of the solution of inventive problems, New York, Gordon and Breeach Science Publ., 1984

[Arm93] Armstrong, James Robert: Structured logic design with VHDL, ISBN 0-13-855206-1, Prentice Hall Verlag, 1993

[Bir80] Birkhofer, H.: Analyse und Synthese der Funktionen technischer Produkte, Dissertation, TU Braunschweig, 1980

[Boe86] Boehm, Barry: A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, August 1986

[Boe88] Boehm, Barry: A Spiral Model of Software Development and Enhancement. IEEE Computer, Vol.21, Ausg. 5, Mai 1988, Seiten 61-72

[Boo96] Booch, Grady: Objektorientierte Analyse und Design, 3., unveränd. Nachdruck, ISBN 3-89319-673-0, Addison-Wesley, Bonn, 1996

[Bru96] Brussel, H. M. J. Van: Mechatronics – A Powerful Concurrent Engineering Framework. IEEE/ASME Transactions on Mechatronics, Vol. 1, No. 2, 1996, S. 127-136

[Cha97] Chang, Henry: A top down, constraint driven design methodology for analog integrated circuits, Kluwer Academic Publishers, Boston/London/Dordrecht, 1997

[CPDD] Counsell, J.; Porter, I.; Dawson, D.; Duffy, M.: Schemebuilder: Computer Aided Conceptual Design, http://www.comp.lancs.ac.uk/edc/schemebuilder/

[CZ03] Czubayko, Roland: Rechnerinterne Repräsentation von informationsverarbeitenden Lösungselementen für die verteilte kooperative Produktentwicklung in der Mechatronik, Dissertation, HNI-Verlagsschriftenreihe, Band 111, ISBN 3-935433-20-4, 2003

[DL93] Dörrscheidt, Frank; Latzel, Wolfgang: Grundlagen der Regelungstechnik, 2., durchges. Aufl., ISBN 3-519-16421-3, Teubner Verlag, Stuttgart, 1993

[Ein00] Ausführungsverordnung zum Gesetz über Einheiten im Messwesen (Einheitenverordnung – EinhV), März 2000. http://www.ptb.de/de/wegweiser/einheiten/si/einhvo.pdf.

[Ehr03] Ehrlenspiel, Klaus: Integrierte Produktentwicklung, 2. Auflage, ISBN 3-446-22119-0, Hanser Verlag, 2003

[ES89] Engels, Gregor; Schäfer, Wilhelm: Programmentwicklungsumgebungen: Konzepte und Realisierung, ISBN 3-519-02487-X, Teubner Verlag, Stuttgart, 1989

[Esc93] Eschermann, Bernhard: Funktionaler Entwurf digitaler Schaltungen, ISBN 3-540-56788-7, Springer Verlag, 1993

[Fla01] Flath, Martin: Methode zur Konzipierung mechatronischer Produkte, Dissertation, ISBN 3-935433-17-4, HNI-Verlagschriftenreihe, Paderborn, 2001

[Föl94] Föllinger, Otto: Regelungstechnik, 8., überarb. Aufl., ISBN 3-7785-2336-8, Hüthig Verlag, Heidelberg, 1994

[GFS05] Gausemeier, Jürgen; Frank, Ursula; Schulz, Bernd: Domänenübergreifende Spezifikation der Prinziplösung selbstoptimierender Systeme unter Berücksichtigung der auf das System wirkenden Einflüsse.; In: Mechatronik 2005, Innovative Produktentwicklung, 2005

[GBF+95] Gausemeier, Jürgen; Brexel, D.;Frank, T.;Humpert, A.: Integrated Product Development – A New Approach for Computer Aided Development in the Early Design Stages, In: Proceedings of the Third Conference on Mechatronics and Robotics, Paderborn, Oktober, 1995

[GEK01] Gausemeier, J.; Ebbesmeyer, P.; Kallmeyer, F.: Produktinnvovation, HANSER – Verlag, ISBN 3-446-21631-6, 2001

[GG97] Grabowski, H.; Geiger, K. (Hrsg.): Neue Wege zur Produktentwicklung. Institut für Rechneranwendung in Planung und Konstruktion, Universität Karlsruhe, Dr. Jodef Raabe Verlags-GmbH, Stuttgart, 1997

[GHJV95] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Design Patterns - Elements of Reusable Object-Oriented Software, ISBN 0-201-63361-2, Addison-Wesley Publishing, 1995

[Gros01] Grosse-Rhode, M.: Integrating Semantics for Object-Oriented System Models. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, Proceedings of ICALP'01, LNCS 2076, S. 40–60. Springer-Verlag, 2001.

[GST04] Gehrke, M.; Steffen, D.; Tichy, M.: Developing an Optimized Principle Solution for Mechatronic Systems, in *Proc. of Mechatronics and Robotics 2004, Aachen, Germany* (P. Drews, ed.), Seiten 1-6, Sascha Eysoldt Verlag, Aachen, September 2004.

[Han74] Hansen, Friedrich: Konstruktionswissenschaft - Grundlagen und Methoden, ISBN 3-446-11957-4, Hanser Verlag, München, 1974

[Hei84] Heinzl, J.: Entwicklungsmethodik für Geräte mit Steuerung durch Mikroelektronik, in: VDI-Berichte Nr. 515, VDI-Verlag, Düsseldorf, 1984

[HTF96] Harashima, F.; Tomizuka, M.; Fukuda, T.: Mechatronics – „What Is It, Why, and How?" An Editorial, IEEE/ASME Transactions on Mechatronics, Vol. 1, No. 1, 1996, S. 1-4

[Hua01] Mei Huang. Funktionsmodellierung und Lösungsfindung mechatronischer Produkte, Dissertation, Universität Karlsruhe, 2001

[Hub93] Huber, Ralf Joachim: Wissensbasierte Funktionsmodellierung als Grundlage zur Gestaltfindung in Konstruktionssystemen, Dissertation, Universität Karlsruhe, Forschungsberichte RPK Band 2, Shaker Verlag, 1993

[iViP02] Krause, F.-L.; Tang, T.; Ahle, U.: Leitprojekt - integrierte Virtuelle Produktentstehung (iViP) - Abschlußbericht, Berlin, 2002

[Ise88] Isermann, R.: Digitale Regelsysteme, Band 1: Grundlagen, Deterministische Regelung, 2. überarbeitete und erweiterte Auflage, Springer Verlag, Berlin et al., 1988

[Ise99] Isermann, R.: Mechatronische Systeme – Grundlagen. Springer Verlag, Berlin et al., 1999

[JAPA71] Japan Trade Mark Kohkoku, Class 9, Shou 46-32713, Januar, 1971

[JAPA72] Japan Trade Registration No. 946594, Januar, 1972

[Kal98] Kallmeyer, Ferdinand: Eine Methode zur Modellierung prinzipieller Lösungen mechatronischer Systeme, Dissertation, Universität Paderborn, ISBN 3-931466-41-8, HNI - Verlagsschriftenreihe, 1998

[KJL04] Krause, Frank-Lothar; Jansen, Helmut; Langenberg, Dirk: Potentiale der Visualisierung für eine verbesserte Produktentstehung, Beitrag zum 100-jährigen Jubiläums des IWF in Berlin, TU Berlin, 2004

[Köc03] Köckerling, Matthias: Methodische Entwicklung und Optimierung der Wirkstruktur mechatronischer Produkte, Dissertation, HNI-Verlagschriftenreihe Nr. 143, Uni Paderborn, Oktober, 2003, ISBN 3-935433-52-2

[Kol98] Koller, Rudolf: Konstruktionslehre für den Maschinenbau, 4. neubearb. und erw. Auflage, ISBN 3-540-63037-6, Springer Verlag, Berlin, 1998

[KU93] Kuttig, Detlef: „Rechnerunterstützte Funktions- und Wirkstrukturverarbeitung beim Konzipieren“, Dissertation, TU-Berlin, Schriftenreihe „Konstrukionstechnik“, Band 25, 1993

[Küs04] Küster, J.: Consistency Management of Object-Oriented Behavioral Models, Dissertation Uni-Paderborn, Paderborn, März 2004

[Lan00] Langlotz, G.: Ein Beitrag zur Funktionsstrukturentwicklung innovativer Produkte, Dissertation, Shaker Verlag, ISBN 3-8265-7246-7, Universität Karlsruhe, 2000

[Lip00] Lippold, Christian: Eine Domänenübergreifende Konzeptionsumgebung für die Entwicklung mechatronischer Systeme, Dissertation, Universität Bochum, Schriftenreihe Heft 00.9, 2000

[LK03] Lückel, J.; Koch, Th.: Funktionsorientierter Entwurf mechatronischer Systeme, Skript zur Vorlesung, Paderborn, 2003

[LKS00] Lückel; Koch; Schmitz: Mechatronik als integrative Basis für innovative Produkte, in: VDI-Tagung: Mechatronik - Mechanisch/Elektrische Antriebstechnik, 29./30. März 2000, Wiesloch; VDI-Verlag, Düsseldorf, 2000

[MC94] Moreira, A. und Clark, R.: Combining Object-Oriented Modeling and Formal Description Techniques. In M. Tokoro and R. Pareschi, editors, Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP’94), S. 344 – 364. LNCS 821, Springer-Verlag, 1994.

[Meh84] Mehlhorn, K.: Graph Algorithms and NP-Completeness, Springer Verlag, 1st Edition, 1984

[Nagl96] Nagl, N. (ed.): Building Tightly Integrated Software Development Environments: The IPSEN Approach, LNCS 1170, Springer, 1996

[NBP] http://nbp-www.upb.de/

[PB03] Pahl, G.; Beitz, W.: Konstruktionslehre – Grundlagen erfolgreicher Produktentwicklung. 5. Auflage, ISBN 3-540-00319-3, Springer Verlag, 2003

[PB96] Plomberger, Gustav; Blaschek, Günther: Software-Engineering, 2., überarb. Aufl., ISBN 3-446-18690-5, Hanser Verlag, München, 1996

[Peri00] Perin, M.: Specification Graphiques Multi-vues: Formalisation et Verification de Coherence. PhD thesis, University of Rennes, Oktober 2000.

[Pur03] Puri, Werner: „Semantische Funktionsmodellierung als Basis für effizientes Product Life Cycle Management“, Dissertation, Uni Erlangen-Nürnberg, 2003, ISBN 3-18-316016-1

[Rot00] Roth, K.: Konstruieren mit Konstruktionskatalogen, 3. Auflage, Springer Verlag, Berlin, ISBN 3540670262, 2001

[Roz97] Rozenberg, G.: Handbook of Graph Grammers and Computing by Graph Transformation, Volume 1, World Scientific Publishing, ISBN 9810228848, 1997

[SCH00] Schön, A.: Konzept und Architektur eines Assistenzsystems für die mechatronische Produktentwicklung, Dissertation, Universität Erlangen-Nürnberg, 2000

[SCH99] Schön, A.: Konzept einer interdisziplinären Effektbibliothek zur Konstruktion mechatronischer Produkte, in: 10. Symposium Fertigungsgerechtes Konstruieren, Erlangen, 1999

[SB96] Samal, Erwin; Becker, Wilhelm: Grundriß der praktischen Regelungstechnik, 19., überarb. und aktualisierte Aufl., ISBN 3-486-23635-0, Oldenbourg Verlag, München, 1996

[SFB01] Sonderforschungsbereich 614, „Selbstoptimierende Systeme des Maschinenbaus“, Finanzierungsantrag 1. Förderperiode, Universität Paderborn, Paderborn, 2001

[SFB04] Sonderforschungsbereich 614, „Selbstoptimierende Systeme des Maschinenbaus“, Finanzierungsantrag 2. Förderperiode, Universität Paderborn, Paderborn, 2004

[Ste01] Stein, Benno Maria: Model Construction in Analysis and Synthesis Tasks, Habilitation, Universität Paderborn, 2001

[SysML04] Systems Modelling Language (SysML) Specification, Version 0.85R1 Draft, Oktober, 2004

[Tei97] Teich, Jürgen: Digitale Hardware/Software Systeme, ISBN 3-540-62433-3, Springer Verlag, Heidelberg, 1997

[TH96] Tomkinson, Donald; Horne, James: Mechatronics Engineering. McGraw-Hill, 1996.

[UML04] OMG: Unified Modeling Language, UML 2.0 Superstructure Specification, Oktober 2004

[Val04] Valckenaers, P.: Challenges of Next Generation Manufactoring Systems, In: Integration of Software Specification Techniques for Applications in Engineering (LNCS 3147), S. 23-28, Springer, 2004, ISBN 3-540-23135-8

[VDI04] VEREIN DEUTSCHER INGENIEURE (VDI): Entwicklungsmethodik für mechatronische Systeme. VDI-Richtlinie 2206, Beuth Verlag, Berlin, 2004

[VDI93] VEREIN DEUTSCHER INGENIEURE (VDI): Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte, VDI-Richtlinie 2221, Beuth Verlag, Berlin, 1993

[VDI94] VEREIN DEUTSCHER INGENIEURE (VDI), VERBAND DEUTSCHER ELEKTROTECHNIKER (VDE): Entwicklungsmethodik für Geräte mit Steuerung durch Mikroelektronik. VDI/VDE-Richtlinie 2422, Beuth Verlag, Berlin, 1994.

[VDI97] VEREIN DEUTSCHER INGENIEURE (VDI): Konstruktionsmethodik - Methodisches Entwickeln von Lösungsprinzipien. VDI-Richtlinie 2222 Blatt 1, Beuth Verlag, Berlin, 1997

[Ver00] Versteegen, Gerhard: Das V-Modell in der Praxis, 1. Aufl., ISBN 3-932588-39-8, dpunkt-Verlag, Heidelberg, 2000

[WLB01] Welp, E.G.; Lippold, C., Bludau, C.: Ein System zur objektorientierten Modellierung mechatronischer Produktkonzepte (ModCoDe), Beitrag zur VDI Mechatronik Tagung, 12./13. September 2001, Frankenthal

[WWW05] http://www-nbp.upb.de/de/konstruktion.php

[Zün02] Zündorf, A.: Rigorous Object Oriented Software Development, Universität Paderborn, 2002, Draft V. 0.3

[ZJ93] Zave, P. und Jackson, M.: Conjunction as Composition. *ACM Transactions on Software Engineering and Methodology*, 2(4): S.379–411, Oktober 1993.

[Zwi71] Zwicky, F.: Entdecken, Erfinden, Forschen im Morphologischen Weltbild, München, Droemer-Knaur 1966/1971