Modellierung von Softwarekomponenten für mechatronische Systeme in UML auf Basis von Systemstrukturen

Matthias Gehrke, Jan Meyer, Wilhelm Schäfer
s-lab
Universität Paderborn
Warburgerstr. 100, 33098 Paderborn
[mgehrke], [janny], [wilhelm]@upb.de

Zusammenfassung

Die durchgängige Modellierung bei der Entwicklung mechatronischer Systeme ist von entscheidender Bedeutung. Im Entwicklungsprozess für mechatronische Systeme nach VDI-Richtlinie 2206 ist dies bisher jedoch nicht hinreichend berücksichtigt. Diese Arbeit zeigt auf, wie eine durchgängige Modellierung auf Basis der VDI-Richtlinie 2206 erreicht werden kann, indem eine Transformation eines Modells des Systementwurfs in ein Modell des domänenspezifischen Entwurfs vorgestellt wird.

Schlüsselwörter

Mechatronische Systeme, Entwicklungsprozess, Modelltransformation

1 Einleitung

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 und der Informatik.

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. 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. 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 [VDI04].

1.1 VDI-Richtlinie 2206

Die VDI-Richtlinie 2206 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 VDI-Richtlinie 2206 basiert auf dem aus der Softwareentwicklung bekannten V-Modell (vgl. Bild 1) und hat dieses gemäß den Anforderungen der Mechatronik angepasst. Es werden grundsätzlich drei Phasen unterschieden: der Systementwurf, der domänenspezifische Entwurf und die Systemintegration.

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

Domänenspezifischer Entwurf: Auf Basis der Prinziplösung werden die zu entwickelnden Elemente auf die jeweiligen Disziplinen verteilt. Hier sollen die etablierten Methoden und Spezifikationstechniken der unterschiedlichen Domänen zum Einsatz kommen.

Systemintegration: Die Systemintegration hat die Aufgabe die in den einzelnen Disziplinen entstandenen Elemente zu einem übergeordneten Ganzen (dem zukünftigen Produkt) zusammenzuführen.

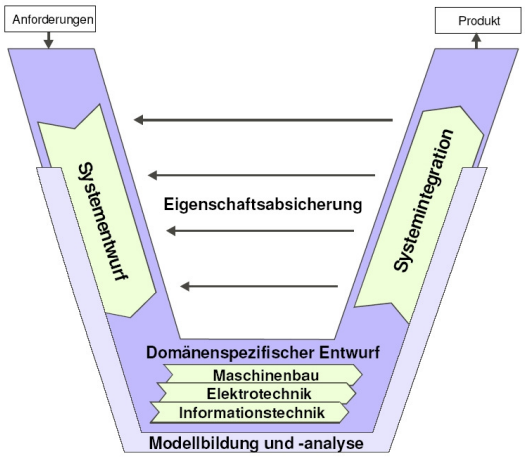

Bild 1: V-Modell der VDI-Richtlinie 2206 [VDI04]

Die VDI-Richtlinie 2206 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.

1.2 Beschreibung der Prinziplösung

Entscheidend für die erfolgreiche Entwicklung mechatronischer Systeme auf Basis der VDI-Richtlinie 2206 ist die enge Verknüpfung der unterschiedlichen Disziplinen. Insbesondere während des *Systementwurfs* ist dies von besonderer Bedeutung, da hier die grundsätzlichen Entscheidungen bzgl. des Systems getroffen werden.

Die Verknüpfung der verschiedenen Disziplinen lässt sich, zumindest während des Systementwurfs, mit Hilfe einer einheitlichen Modellierung erreichen. Hierzu werden Spezifikationstechniken benötigt, die das zu entwickelnde mechatronische System (Struktur und Verhalten) beschreiben und von den Mitarbeitern der verschiedenen Disziplinen verstanden werden. Eine Möglichkeit hierzu wurde von Ursula Frank [Fra06] erarbeitet, indem sie mehrere Spezifikationstechniken definiert hat, die in ihrer Gesamtheit die Prinziplösung (vgl. Bild 2) beschreiben. Diese Spezifikationstechniken stehen untereinander in Verbindung bzw. bauen aufeinander auf und dienen als Vorgabe für die dann folgende Phase, dem *domänenspezifischen Entwurf*.

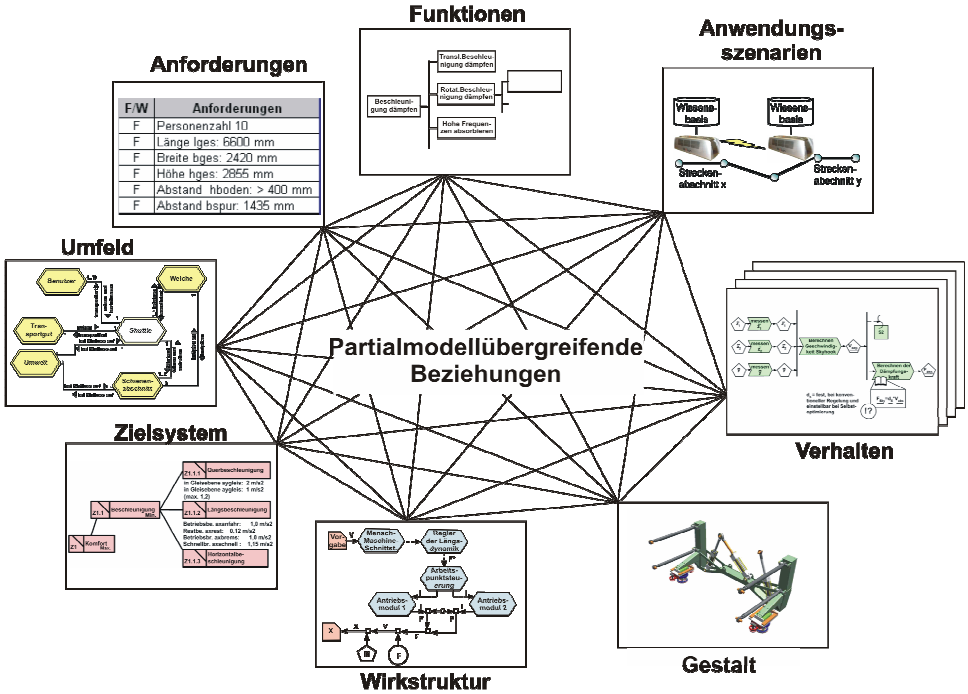

Bild 2: Spezifikationstechniken der prinzipiellen Lösung [Fra06]

1.3 Mangelnde Verknüpfung der Spezifikationstechniken

Die VDI-Richtlinie 2206 und die Arbeit von Ursula Frank bringen eine Vielzahl von Verbesserungen mit sich. Jedoch bleiben noch einige Probleme nach wie vor ungelöst. So wird z.B. in der VDI-Richtlinie nicht festgelegt, welche Spezifikationstechniken in der Phase des *domänenspezifischen Entwurfs* eingesetzt werden sollen. Dies können in der Disziplin Informatik beispielsweise verschiedene Spezifikationstechniken der Unified Modeling Language (UML), einem Standard in dieser Disziplin, sein. In der Regelungstechnik können dies Blockschaltbilder sein und in der Mechanik CAD-Zeichnungen.

Das Problem liegt nun darin, in den Spezifikationstechniken der Prinziplösung diejenigen Informationen zu ermitteln, die in den Spezifikationstechniken des domänenspezifischen Entwurfs benötigt bzw. weiter verwendet werden können. Diese Informationen müssten idealerweise automatisch ermittelt und in die entsprechenden Spezifikationstechniken automatisch übertragen werden. Auf diese Weise ließe sich der Entwicklungsprozess beschleunigen und die Fehlerwahrscheinlichkeit, gegenüber einer Transformation „von Hand“, deutlich reduzieren.

2 Transformationsalgorithmus

In diesem Kapitel werden die beschriebenen Probleme aufgegriffen und eine Lösung vorgestellt, die eine Transformation zweier Spezifikationstechniken beschreibt, die in den unterschiedlichen Phasen des Entwicklungsprozesses eingesetzt werden (vgl. Bild 3). Hierbei handelt es sich zum einen um die Systemstruktur, die eine Spezifikationstechnik der prinzipiellen Lösung ist und zum anderen um das UML-Komponentendiagramm, welches im *domänenspezifischen Entwurf* in der Disziplin Informatik verwendet wird.

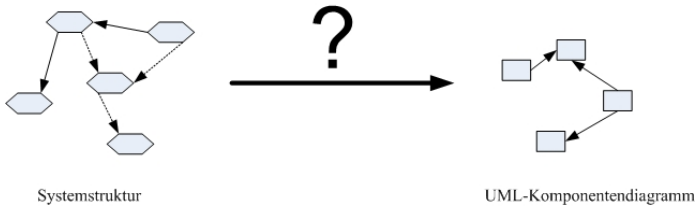

Bild 3:Bild von der Systemstruktur zum UML-Komponentendiagramm

Die Verwendung einer Transformation ist in diesem Fall ein sinnvoller Weg, da andere Lösungsmöglichkeiten zu komplex sind. So wäre es theoretisch auch möglich ein gemeinsames Meta-Modell für alle Modelle zu entwerfen, wie dies z.B. im Projekt MECHASOFT [ALR05] praktiziert wird. In diesem Meta-Modell müssten dann jedoch alle Spezifikationstechniken sowohl der prinzipiellen Lö-

sung, als auch der des domänenspezifischen Entwurfs enthalten sein. Dies würde allerdings zu einem sehr umfangreichen und unübersichtlichen Meta-Modell führen. Darüber hinaus werden die Werkzeuge zur Modellierung der Modelle von unterschiedlichen Herstellern entwickelt, was die Nutzung eines gemeinsamen Meta-Modells zusätzlich erschweren würde.

2.1 UML-Komponentendiagramm

UML-Komponentendiagramme werden zur Modellierung der Struktur eines Softwaresystems innerhalb der Disziplin Informatik verwendet. Dies geschieht durch die Aufteilung des Systems in verschiedene Komponenten. Eine Komponente kapselt dabei ein komplexes Verhalten und stellt eine ausführbare und austauschbare Softwareeinheit dar [Oes01]. Die Kommunikation mit anderen Komponenten erfolgt über Schnittstellen (Ports).

Bild 4 zeigt beispielhaft ein UML-Komponentendiagramm einer Geschwindigkeitsregelung aus dem Projekt der „Neuen Bahntechnik Paderborn“ [NPB06]. Das Diagramm zeigt zwei verschiedenen Komponenten (Regler), die je nach Bedarf abwechselnd aktiv sind. Welche Komponente jeweils aktiv ist, ist abhängig vom Abstand zu einem vorausfahrenden Shuttle. Dieser Abstand wird mittels eines Abstandssensors an die Komponenten innerhalb der Geschwindigkeitssteuerung weitergegeben. Bei einem sehr geringen Abstand übernimmt der Abstandsregler die Steuerung, während bei einem größeren Abstand, falls es z.B. kein vorausfahrendes Shuttle gibt, der Geschwindigkeitsregler aktiv ist. Wichtig ist, dass es sich bei allen Komponenten um reine Softwarekomponenten handelt.

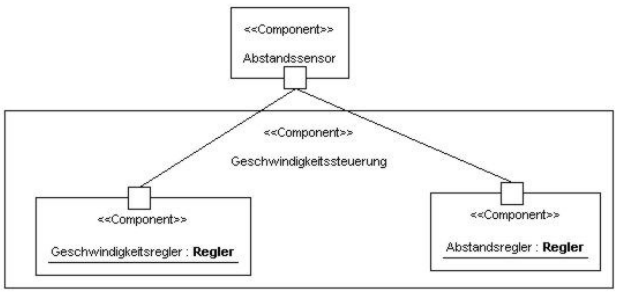

Bild 4: UML-Komponentendiagramm der Geschwindigkeitssteuerung

2.2 Die Systemstruktur

Die Systemstruktur ist eine Spezifikationstechnik zur Beschreibung der Struktur eines mechatronischen Systems. Sie wurde von Ferdinand Kallmeyer [Kal98]

entwickelt und ist eine der Spezifikationstechniken der prinzipiellen Lösung (vgl. Bild 2).

Die Systemstruktur besteht aus Systemelementen, die die verschiedenen Komponenten beschreibt, aus denen das mechatronische System aufgebaut wird, wobei die Komponenten verschiedenen Disziplinen zugeordnet werden können. Es werden sowohl Hardware-Komponenten (z.B. Steuergeräte), als auch Software-Komponenten (z.B. Regler) gemeinsam beschrieben. Die Kommunikation zwischen den Systemelementen wird mit Hilfe von Flüssen modelliert, wobei es drei verschiedene Arten von Flüssen gibt, Energie-, Stoff- und Informationsflüsse.

In Bild 5 ist beispielhaft eine Systemstruktur dargestellt. Sie besteht aus fünf verschiedenen Systemelementen und stellt ebenfalls die Geschwindigkeitssteuerung eines Shuttles dar (vgl. Kapitel 2.1).

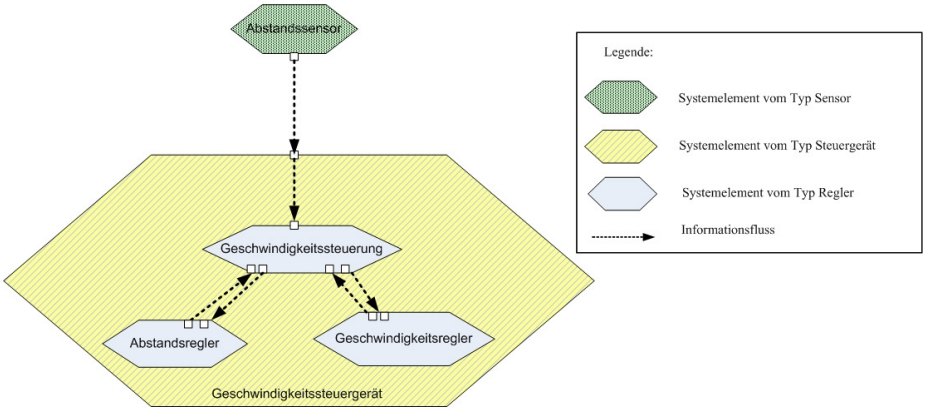

Bild 5: Systemstruktur der Geschwindigkeitsteuerung

Wie zu erkennen ist, ähneln sich die beiden Spezifikationstechniken sehr stark und daher liegt es nahe, dass Systemelemente aus der Systemstruktur im UML-Komponentendiagramm weiterverwendet werden. Damit dies jedoch möglich ist, muss eine entsprechende Transformation zwischen den Spezifikationstechniken definiert werden. Diese Transformation lässt sich allerdings nur dann definieren, wenn zuvor einige Schwachstellen behoben werden.

Hier ist insbesondere die fehlende Formalisierung der Systemstruktur zu nennen. Kallmeyer erläutert zwar den Aufbau der Systemstruktur, also die graphischen Konstrukte, doch wird keine Formalisierung z.B. in Form eines Meta-Modells angegeben. Dies ist jedoch notwendig, um die gewünschte Transformation formal beschreiben zu können.

2.3 Formalisierung der Systemstruktur

Im Rahmen der hier vorgestellten Arbeit, wurde die Systemstruktur von Kallmeyer ergänzt bzw. angepasst. Insbesondere wurde die Systemstruktur formalisiert, indem das in Bild 6 gezeigte Meta-Modell definiert wurde.

Das entwickelte Meta-Modell besteht aus fünf Klassen. In der Klasse *MDEModel* werden die Systemelemente und die Flüsse der Systemstruktur gespeichert. Dabei werden die Flüsse durch die Klasse *Connection* im Meta-Modell repräsentiert. Sie benötigen jeweils einen Start- als auch einen Endport an den jeweiligen Systemelementen. Diese Ports werden durch die Klasse *ConnectionAnchor* dargestellt. Die Klasse *SolutionNode* im Meta-Modell repräsentiert ein Systemelement der Systemstruktur. In dieser Klasse werden alle Eigenschaften des Systemelementes gespeichert. Dies ist sowohl der Typ des Systemelementes, der in der Variablen *style* gespeichert wird, als auch Eigenschaften, die mittels der Klasse *Attribute* beschrieben werden.

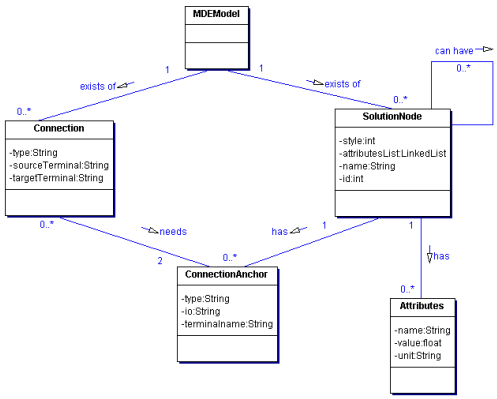

Bild 6: Meta-Modell der Systemstruktur als UML-Klassendiagramm

2.4 Transformationsregeln

Im Folgenden wird gezeigt, wie die Transformation von der Systemstruktur zum UML-Komponentendiagramm formalisiert wurde. Zu diesem Zweck wurden Transformationsregeln definiert, die festlegen, welche Elemente der Systemstruktur auf welche Elemente in einem UML-Komponentendiagramm abgebildet werden. Die Beschreibung der Transformationsregeln wurde mit Hilfe der Queury/View/Transformation (QVT) [QVT06] vorgenommen. QVT ist Teil der UML 2.0 Spezifikation und ermöglicht die formale Beschreibung von Transformationen.

Die Transformationsregeln beruhen auf den jeweiligen Meta-Modellen der beiden Spezifikationstechniken. Das Meta-Modell der Systemstruktur ist in Kapitel 2.3

beschrieben. Das Meta-Modell des UML-Komponentendiagramm ist ähnlich und wurde daher aus Platzgründen weggelassen.

Regel 1: Diese Regel bildet ein Systemelement (*SolutionNode*) auf eine Komponente (*Component*) ab (vgl. Bild 7). Dabei werden der *name* und die *id* übernommen. Dies ist in Bild 8 daran zu erkennen, dass bei beiden Domänen dieselbe Variable verwendet wird (pn, cn). Die Transformationsregel wird allerdings nur dann angewendet, wenn der Typ (*style*) für die Modelltransformation vorgesehen ist, d.h. das Systemelement auch im UML-Komponentendiagramm als Komponente verfügbar sein soll.

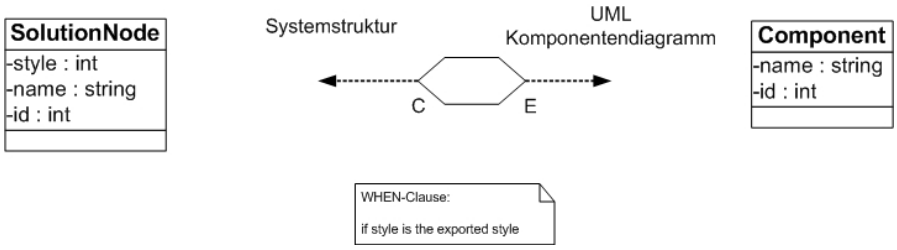

Bild 7: Transformation eines Systemelementes beschrieben in QVT

```
Relation SolutionNodeToComponent
Checkonly domain systemstruktur n:Node {
Name = pn,
Id = cn,
Style = styleType}
Enforce domain uml c:Component{
Name =pn,
Id = cn}
```

Bild 8: Auszug der textuellen Transformationsbeschreibung

Regel 2: Diese Regel bildet die Ports der Systemelemente (*ConnectionAnchor*) auf Ports der Komponenten (*Port*) ab (vgl. Bild 9). Hierbei wird der *terminalname* abgebildet auf die Variable *name*. Die Variable *io* wird übernommen.

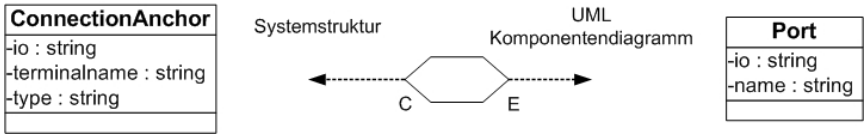

Bild 9: Transformation eines Ports beschrieben in QVT

Regel 3: Diese Regel bildet die Flüsse der Systemstruktur (*Connection*) auf die Verbindungen zwischen den Komponenten (*Delegation*) ab (vgl. Bild 10). Bei

dieser Transformation werden die Start- und Endports der Systemelemente (*source-* bzw. *targetTerminalname*) zu Ports (*startPort* und *endPort*) der Komponenten umgewandelt.

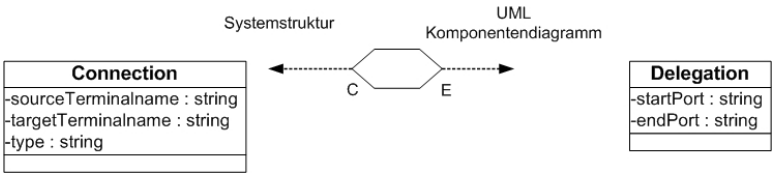

Bild 10: Transformation eines Flusses beschrieben in QVT

2.5 Prototypische Realisierung

Um eine Werkzeugunterstützung (vgl. Bild 11) anbieten zu können, wurde ein Prototyp in Form eines Eclipse-Plugins implementiert, der die definierten Transformationsregeln anwendet. Dieser Prototyp ist in der Lage Systemelemente einer Systemstruktur in ein UML-Komponentendiagramm des Softwarewerkzeugs Fujaba zu transformieren.

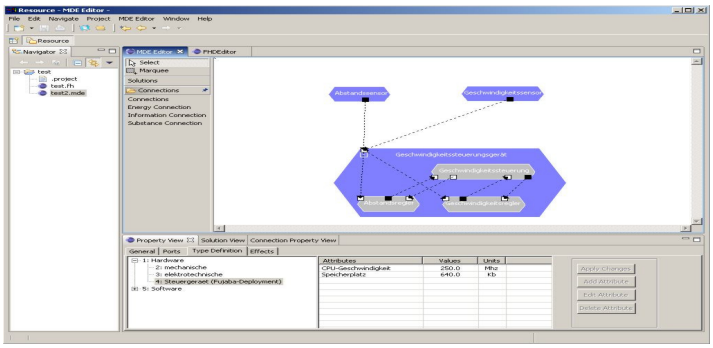

Bild 11: Systemstruktureditor

3 Zusammenfassung und Ausblick

Die vorgestellte Arbeit beschäftigt sich mit dem Entwicklungsprozess mechatronischer Systeme auf Basis der VDI-Richtlinie 2206. Dabei ist das in der Richtlinie vorgeschlagene V-Modell zunächst erläutert und dessen Schwachstellen herausgearbeitet worden. Zu diesen Schwachstellen gehört insbesondere der nur sehr informal beschriebene Austausch von Modellinformationen zwischen den, während der Entwicklung verwendeten Spezifikationstechniken.

Diese Problematik wurde aufgegriffen und eine entsprechende Lösung erarbeitet. Dabei wurde die Systemstruktur zunächst formalisiert. Hierauf aufbauend wurde eine Modelltransformation erarbeitet, die Systemelemente der Systemstruktur in

Komponenten eines UML-Komponentendiagramms transformiert. So können die Daten aus der Phase des Systementwurfs auch im weiteren Verlauf der Entwicklung automatisch weiter genutzt werden. Schließlich wurde ein Prototyp entwickelt, dar die Transformationsregeln anwendet.

Der beschriebene Ansatz beschreibt allerdings nur die Transformation zwischen zwei Spezifikationstechniken. Während der Entwicklung gibt es noch eine Vielzahl anderer Spezifikationstechniken. Auch für diese Spezifikationstechniken besteht die Möglichkeit entsprechende Transformationsregeln anzugeben. Dies hat dann allerdings zur Folge, dass ein gesamtheitlicher Integrationsansatz benötigt wird. Es müssen alle Modelle der Prinziplösung untereinander verknüpft werden. Ein Weiteres, hier nicht betrachtetes Problem ist die Konsistenzhaltung der Modelle. Bei Änderungen in einem Modell muss sichergestellt werden, dass diese Änderungen auch in anderen Modellen nachgezogen werden. Einen Ansatz hierzu beschreibt M. Gehrke [Geh05], indem er die Konsistenz zweier Modelle der prinzipielle Lösung (Funktionshierarchie und Systemstruktur) zueinander konsistent hält.

Literatur

[ALR05] Anton, O., Lercher, B und Reinhart, G.: Funktionsorientiertes Sichtenmodell für die Entwicklung mechatronischer Systeme, VDI-Z, München, 2005

[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

[Fra06] Frank, Ursula: Spezifikationstechnik zur Beschreibung der Prinziplösung selbstoptimierender Systeme. Dissertation, Universität Paderborn, Heinz Nixdorf Institut, HNI-Verlagsschriftenreihe, Paderborn, Band 175, 2006

[Geh05] Gehrke, M.: Entwurf mechatronischer Systeme auf Basis von Funktionshierarchien und Systemstrukturen, Dissertation, Oktober 2005, Universität Paderborn

[Oes01] Oestereich, B.: Objekt-orientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language, Oldenbourg Verlag, 5. Edition, 2001

[Kal98] Kallmeyer, F. : Eine Methode zur Modellierung prinzipieller Lösungen mechatronischer Systeme. Dissertation, Universität Paderborn, Heinz Nixdorf Institut, HNI-Verlagsschriftenreihe, Paderborn, Band 42, 1998.

[NPB06] Neue Bahntechnik Paderborn, www.npb.de, 2006

[QVT06] QVT-Spezifikation, www.omg.org/docs/ad/02-04-10.pdf, 2006

[VDI04] VEREIN DEUTSCHER INGENIEURE (VDI): Entwicklungsmethodik für mechatronische Systeme. VDI-Richtlinie 2206, Beuth Verlag, Berlin, 2004