Modellkonsistenz-Management im Systems Engineering[1]

Jan Dominik Rieke[2]

Abstract: Um der Komplexität der interdisziplinären Entwicklung moderner technischer Systeme Herr zu werden, findet die Entwicklung heutzutage meist modellbasiert statt. Dabei werden zahlreiche verschiedene Modelle genutzt, die jeweils unterschiedliche Gesichtspunkte berücksichtigen und sich auf verschiedenen Abstraktionsebenen befinden. Wenn die hierbei auftretenden Inkonsistenzen zwischen den Modellen ungelöst bleiben, kann dies zu Fehlern im fertigen System führen.
Modelltransformations- und -synchronisationstechniken sind ein vielversprechender Ansatz, um solche Inkonsistenzen zu erkennen und aufzulösen. Existierende Modellsynchronisationstechniken sind allerdings nicht mächtig genug, um die komplexen Beziehungen in so einem Entwicklungsszenario zu unterstützen. In dieser Arbeit wird eine neue Modellsynchronisationstechnik präsentiert, die es erlaubt, Modelle verschiedener Sichten und Abstraktionsebenen zu synchronisieren. Dabei werden Metriken zur Erhöhung des Automatisierungsgrads eingesetzt, die Expertenwissen abbilden. Der Ansatz erlaubt unterschiedliche Grade an Benutzerinteraktion, von vollautomatischer Funktionsweise bis zu feingranularen manuellen Entscheidungen.

1 Einleitung

Angefangen von Haushaltsgeräten bis hin zu Transportmitteln – moderne technische Systeme werden immer komplexer. Sie beinhalten zudem einen stetig größer werdenden Anteil an Software. Software spielt eine Schlüsselrolle insbesondere bei untereinander stark vernetzten Systemen (sogenannte „Systems of Systems“). Oft werden solche Systeme auch mit dem Begriff „Mechatronik“ beschrieben. Dieser Begriff beschreibt das Zusammenspiel und die Kollaboration verschiedener Fachdisziplinen wie Mechanik, Elektrotechnik/Elektronik, Regelungstechnik und Softwaretechnik.

Die steigende Komplexität stellt die Entwicklung solcher Systeme vor Herausforderungen. Bisher sind die fachdisziplinspezifischen Aspekte der Entwicklung meist nur wenig bis gar nicht in den Gesamt-Entwicklungsprozess integriert. Zudem sind die Entwicklungsprozesse der einzelnen Fachdisziplinen nur unzureichend aufeinander abgestimmt. Insbesondere erfordert die Entwicklung komplexer technischer Systeme einen Blick auf das „System als Ganzes“, der alle disziplinübergreifend relevanten Aspekte der Entwicklung berücksichtigt und in dem alle Fachdisziplinen berücksichtigt sind. Dies wird auch als *Systems Engineering* bezeichnet. Systems Engineering bezieht sich dabei sowohl auf das zu entwickelnde System als auch auf den zugrundeliegenden Entwicklungsprozess: „Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation.“ [In14]

[1] Englischer Titel der Dissertation: Model Consistency Management for Systems Engineering

[2] Fachgruppe Softwaretechnik, Heinz Nixdorf Institut, Universität Paderborn, info@janrieke.de

Es gibt zahlreiche Design-Richtlinien, die die Entwicklung verschiedener System-Typen adressieren. Abb. 1 zeigt das *V-Model*, ein typisches Prozessmodell für die Entwicklung technischer Systeme [Be05, Bu06]. Für solche technischen bzw. mechatronischen Systeme schlägt die VDI 2206 [Ve04] oder eben dieses V-Modell [Be05, Bu06]) vor, dass die Entwicklung in drei Phasen stattfindet.

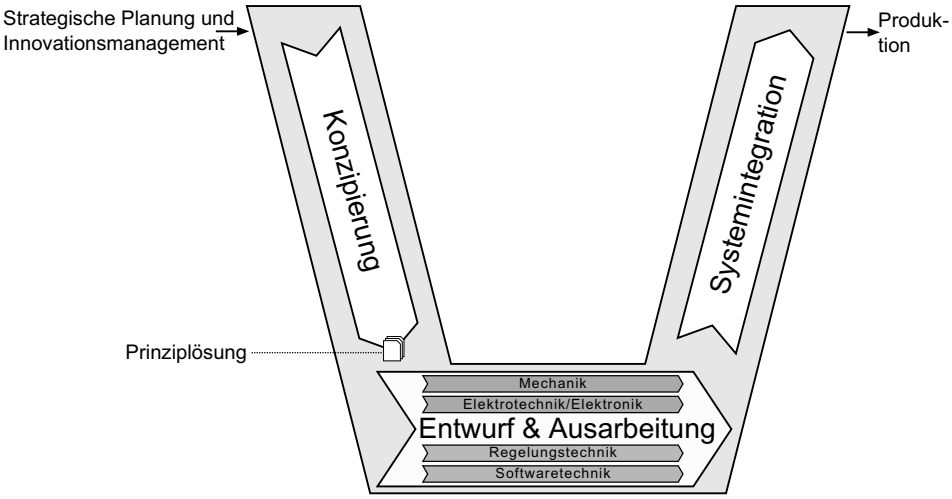

Abb. 1: V-Modell als Prozessmodell für die Entwicklung mechatronischer Systeme (basierend auf [Ve04, Bu06])

Nach der initialen strategischen Produktplanung beginnt die Entwicklung zunächst mit der Phase *Konzipierung*. Dabei kollaborieren Experten aller Disziplinen und erarbeiten die sogenannte *Prinziplösung*, ein Modell des zu entwickelnden Systems, das alle fachdisziplinübergreifend relevanten Informationen beinhaltet. Diese Prinziplösung dient anschließend als Ausgangsbasis für die Phase *Entwurf und Ausarbeitung*. Darin entwickeln die einzelnen Fachdisziplinen die Einzelheiten des Systems aus Sicht ihrer Disziplin mit Hilfe ihrer eigenen Entwicklungsartefakte und Werkzeuge. Die grundsätzliche Idee dabei ist, dass Entwicklungsaufgaben, die bislang konsekutiv erfolgten, jetzt stark parallelisiert durchgeführt werden. Diese Idee ist auch bekannt als „Concurrent Engineering“ [Fo95, MCT08]. Abschließend werden die Ergebnisse in der Phase *Systemintegration* zusammengeführt: Die verschiedenen Entwicklungsartefakte der einzelnen Fachdisziplinen werden zu einem Gesamtsystemmodell kombiniert. Dieses kann dann beispielsweise für modellbasiertes Testen oder zu Simulationszwecken genutzt werden und dient anschließend als Basis für die Produktion.

Das V-Modell selbst ist lediglich ein organisatorisches Rahmenwerk für den Ablauf der Entwicklung. Es muss daher noch ausgestaltet werden für das gerade zu entwickelnde System. Außerdem enthält es keine Informationen darüber, wie die Abhängigkeiten der Fachdisziplinen untereinander sind. Daher wurde im Rahmen des SFB 614 „Selbstoptimierende Systeme des Maschinenbaus“ eine Entwicklungsmethodik entwickelt, die speziell auf mechatronische Systeme zugeschnitten ist (siehe auch GAUSEMEIER ET AL. [Ga14]).

Ein Teil dieser Methodik ist die *Kollaboration und Koordination der Fachdisziplinen*. Aus technischer Sicht spielt dabei die Sicherstellung der *Konsistenz der verschiedenen Entwicklungsartefakte* während des gesamten Entwicklungsprozesses eine Schlüsselrolle.

Idealerweise beschreibt die Prinziplösung alle interdisziplinär relevanten Informationen abschließend. In so einem Fall gäbe es keine Notwendigkeit für disziplinübergreifende Abstimmungen während der Ausarbeitungsphase. In der Praxis zeigt sich jedoch, dass regelmäßig während dieser zweiten Entwicklungsphase disziplinübergreifende Änderungen notwendig werden, die in der Konzipierungsphase nicht vorhersehbar waren. So könnten neue Abhängigkeiten zwischen den Disziplinen entdeckt werden, oder Änderungen an der Prinziplösung werden notwendig aufgrund von sich ändernden Anforderungen.

Solcherlei Probleme sind lange bekannt (vgl. z. B. FOHN ET AL. [Fo95]). Laut MA ET AL. ist es im Concurrent Engineering „nicht einfach, die Konsistenz zwischen zusammenhängenden Produktmodellen zu gewährleisten, da die Verbindungen der Informationen nicht hergestellt sind" [MCT08] (Übersetzung vom Autor). Diese Probleme sind bis heute nur unzureichend gelöst. Erforderlich ist eine Methode, die disziplinübergreifend relevante Änderungen identifiziert und sie in andere Fachdisziplinen weiterleitet. Andernfalls kann es passieren, dass die entwickelten Artefakte nicht zueinander passen und so ein fehlerhaftes Gesamtsystem entsteht.

Um der Komplexität der interdisziplinären Entwicklung moderner technischer Systeme Herr zu werden, findet die Entwicklung heutzutage meist *modellbasiert* statt. Laut STACHOWIAK [St73] ist ein Modell eine verkürzte (*abstrahierende*) *Abbildung* von Entitäten der Wirklichkeit und Beziehungen zwischen diese Entitäten, die für einen bestimmten Zweck genutzt wird (*Pragmatismus*). Im Konstruktionswesen vereinfachen Modelle die Entwicklung, indem z. B. von unwichtigen Details abstrahiert wird und die Modelle so das Verständnis bestimmter Systemaspekte fördern. In der rechnergestützten Entwicklung (engl. „computer-aided engineering", kurz: *CAE*) existieren Modelle in der Regel nur als Repräsentation in Computern.

Während der Entwicklung werden in allen Phasen zahlreiche verschiedene Modelle genutzt, die jeweils unterschiedliche Aspekte des Systems darstellen. Sie unterscheiden sich in Detailgrad, Gesichtspunkt und Zweck. Es gibt beispielsweise Modelle, die ausschließlich Aspekte einer Disziplin abdecken, und fachdisziplinübergreifende Modelle wie das Systemmodell.

Wegen der unterschiedlichen Abstraktion und Abbildung gleicher Sachverhalte konstituieren sich Änderungen in unterschiedlicher Weise in den Modellen. Zudem können beteiligte Ingenieure aufgrund der Unterschiedlichkeit der Modelle die Konsequenzen einer Änderung auf andere Modelle kaum überblicken. Automatische Modelltransformations- und -synchronisationstechniken sind ein vielversprechender Ansatz in solchen Szenarien, um Modelle auch bei Änderungen konsistent zu halten. Bisherige Ansätze sind jedoch nicht anwendbar, da sie sich meist auf einfache Transformationsszenarien fokussieren oder bestimmte Funktionen nicht bieten, die aufgrund der unterschiedlichen Abstraktionsebenen zwischen den Modellen nötig wären.

Zusammenfassend lässt sich sagen, dass das beschriebene Szenario Modelltransformations- und -synchronisationstechniken vor neue Herausforderungen stellt: Wir haben es mit einer Vielzahl verschiedenartiger Modelle zu tun, deren Abbildung aufeinander ein kompliziertes Unterfangen darstellt. Zudem entsteht aufgrund der komplexen Beziehungen zwischen den Modellen eine große Menge an Abbildungsvorschriften, deren Definition und Wartung allein aufgrund ihrer Größe sehr schwer ist. Zudem sind (aufgrund der unterschiedlichen Abstraktionsebenen) Abbildungsvorschriften in der Regel mehrdeutig.

Bevor wir auf die im Rahmen der Dissertation entwickelten Methoden eingehen, soll zunächst ein Beispielszenario skizziert werden, das wir im weiteren Verlauf nutzen werden.

2 Beispielszenario

Als fortlaufendes Beispiel dient das Forschungsprojekt „RailCab – Neue Bahntechnik Paderborn“[3]. Die Vision dieses Projekts ist ein fahrerloses, schienengebundenes Transportsystem, das nicht mehr fahrplangebunden ist, sondern stattdessen auf Basis individueller Nachfrage funktioniert. Die Idee ist, dass der bestehende Schienenverkehr um kleine, autonom fahrende „RailCabs“ (übersetzt in etwa „Schienentaxis“) ergänzt wird, die Passagiere und Güter auf Anforderung transportieren. Während der Fahrt sollen diese RailCabs dynamisch kontaktlose Konvois bilden, um Energie zu sparen. Abb. 2 illustriert diese Vision und zeigt unter anderem die Testfahrzeuge und die Teststrecke an der Universität Paderborn.

Konvoi-Bildung

Teststrecke und Testfahrzeuge an der Universität Paderborn

Abb. 2: Das Forschungsprojekt „RailCab – Neue Bahntechnik Paderborn“

Im Folgenden wollen wir näher betrachten, wie ein modellbasierter Entwicklungsprozess auf dieses RailCab-Projekt angewendet werden könnte. In der ersten Phase entwickelt ein interdisziplinäres Ingenieursteam die Prinzipiösung des RailCab-Systems. Dabei muss unter anderem festgelegt werden, wie die verschiedenen RailCabs miteinander kommunizieren. Dazu wird ein Protokoll definiert, über das die RailCabs untereinander die Bildung von Konvois koordinieren können. Dies erfolgt typischerweise mit Zustandsdiagrammen.

Befindet sich ein RailCab allein auf freier Strecke, so ist es im Zustand NoConvoy und bestimmt seine Fahrtgeschwindigkeit selbst. Fährt es allerdings zusammen mit anderen RailCabs in einem Konvoi (Zustand Convoy), so muss es seine Fahrtgeschwindigkeit an die Geschwindigkeit der anderen RailCabs anpassen. Folglich sind für diese zwei Mo-

[3] Webseite: `http://www-nbp.uni-paderborn.de/`

di unterschiedliche Geschwindigkeitsregler notwendig. Die Umschaltung zwischen den Reglern erfolgt genau dann, wenn eine Zustandsänderung passiert.

Die Details der Geschwindigkeitsregler und des Umschaltvorgangs werden dabei in der zweiten Phase Entwurf und Ausarbeitung von Ingenieuren der Regelungstechnik festgelegt. Die Softwaretechnik ist dafür zuständig, die Kommunikation und die Protokolle zu umzusetzen. Änderungen am Protokoll durch Softwaretechnik-Ingenieure können folglich Auswirkungen auf die Regelungstechnik haben, da dort die Reglerumschaltung von den Zustandswechseln ausgelöst wird.

Abb. 3 zeigt, wie sich die verschiedenen Modelle im beschriebenen Beispielszenario entwickeln und wie Informationen zwischen den Modellen propagiert werden. Nach der Generierung initialer fachdisziplinspezifischer Modelle aus der Prinzipiösung in Schritt 1 beginnen die Ingenieure der Disziplinen mit der Ausarbeitung ihrer Modelle. Z. B. werden Regelungsstrategien durch die Regelungstechnik implementiert (Schritt 2). In Schritt 3 wird nun durch die Softwaretechnik eine Änderung am Protokoll getätigt. Diese Änderung muss nun an alle disziplinspezifischen Modelle weitergeleitet werden (Schritt 4).

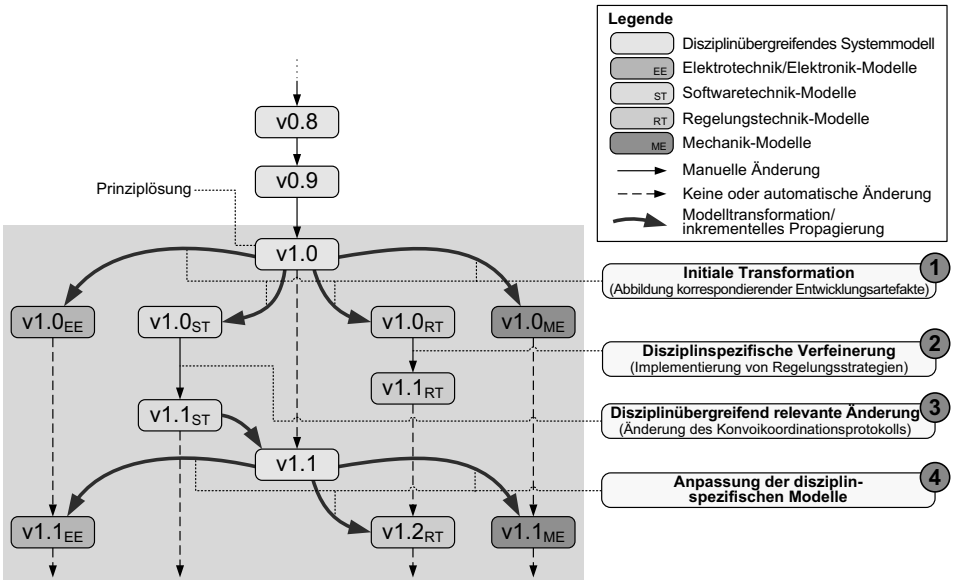

Abb. 3: Entwicklung der verschiedenen Modelle im Beispielszenario

3 Problem

An Algorithmen, Sprachen und Werkzeugen für bidirektionale Modelltransformation und -synchronisation wird heute vielfach geforscht (vgl. z. B. [HLR06, GH09, GW09, Xi09, La12, Hi12, La13]). Nichtsdestotrotz zielen existierende Ansätze vor allem auf Anwendungsszenarien ab, bei denen Modelle ähnlicher Ausdrucksmächtigkeit oder strukturell

ähnliche Modelle konsistent gehalten werden sollen [RS12]. Wenn – wie in unserem Fall – Modelle unterschiedlicher Abstraktionsebenen oder Blickwinkel synchronisiert werden sollen (sogenannte „vertikale Transformationen“[4]), sind bisherige Ansätze unzureichend.

Bestehende Ansätze haben dabei insbesondere drei Probleme in unserem Szenario:

1. Sie verursachen unnötigen *Informationsverlust*. Als die Softwaretechnik das Protokoll ändert, hat die Umsetzung der Geschwindigkeitsregler der Regelungstechnik bereits begonnen – die Modelle enthalten nun disziplinspezifische Informationen. Im Beispielszenario in Abb. 3 ist dies der Fall beim Regelungstechnikmodell ($v1.1_{RT}$). Diese zusätzlichen Informationen sind nicht im disziplinübergreifenden Systemmodell enthalten; sie sind also nicht Teil der Modelltransformation. Wenn so ein angereichertes Modell durch existierende Modelltransformationstechniken geändert würde, käme es unweigerlich zum Verlust zumindest eines Teils der disziplinspezifischen Informationen – die Transformation hat keine Kenntnis von der Existenz dieser Informationen und kann sie somit leicht beschädigen oder löschen.
2. Bestehende Ansätze bieten keine ausreichende Unterstützung für Transformationen zwischen Modellen *unterschiedlicher Abstraktionsebenen*. In unserem Fall müssen abstraktere Modelle wie das Systemmodell mit detaillierten Fachdisziplin-Modellen synchronisiert werden. Die detaillierten Modelle enthalten zu bestimmten Aspekten mehr Detail-Informationen; daher gibt es oft mehrere Möglichkeiten, ein Konstrukt aus dem abstrakten Modell ins detaillierte zu übersetzen.
3. Während des hochgradig parallelen Entwicklungsprozesses können *Bearbeitungskonflikte* entstehen, wenn gleichzeitig zusammengehörige Teile von zwei Modellen widersprüchlich verändert werden. Die Modellsynchronisation muss hier Unterstützung bieten, um möglichst viele dieser Konflikte automatisch zu beheben. Gleichzeitig muss sie auch Unterstützung bieten für manuelle Konfliktauflösung, falls Konflikte nicht automatisch lösbar sind.

Nicht zuletzt leiden bisherige Ansätze unter geringer Verständlichkeit und Wartbarkeit, sobald die Transformationsszenarien größer werden.

4 Beiträge der Arbeit

Im Kern der Arbeit steht ein dreiteiliger Ansatz, der die zuvor beschriebenen Probleme angeht. Er wird ergänzt durch Techniken zur Vereinfachung und Verbesserung der Entwicklung von Transformationen und deren Wartbarkeit.

Als zugrundeliegenden Modelltransformationsformalismus nutzen wir Tripel-Graph-Grammatiken (TGGs). TGGs sind eine regelbasierte, deklarative Modelltransformationssprache, die 1994 von SCHÜRR [Sc95] eingeführt wurde. Die Semantik ist präzise definiert

[4] Horizontale Transformationen bilden Modelle gleichen Abstraktionsgrads aufeinander ab, vertikale Transformationen Modelle unterschiedlicher Abstraktionsgrade [MG06].

und basiert auf dem etablierten Konzept der Graphgrammatiken. Dies erlaubt beispielsweise eine formale Analyse und Verifikation; so kann die Korrektheit von Transformationsergebnissen garantiert werden. Dies ist gerade bei der Entwicklung sicherheitskritischer Systeme wichtig.

4.1 Intelligente Propagierung von Änderungen

Um Informationsverlust vorzubeugen, wird ein neuer Synchronisationsalgorithmus genutzt, der bei der Aktualisierung eines Zielmodells die Auswirkungen auf disziplinspezifische Informationen zu minimieren versucht. Die zugrundeliegende Idee ist hier, bei der Löschung von Modellelementen (verursacht durch Regelrücknahmen) diese nicht direkt zu entfernen, sondern sie zunächst nur als *zum Löschen vorgesehen* zu markieren. Die so markierten Modellelemente werden anschließend bei der Regelneuanwendung intelligent wiedergenutzt, sofern dies möglich ist.

Dabei kann es vorkommen, dass mehrere (im Sinne der Transformation korrekte) Möglichkeiten zur Wiederverwendung existieren. In diesem Fall werden Heuristiken auf Basis von Metriken eingesetzt, um die wahrscheinlich sinnvollste Möglichkeit zu identifizieren. Die Grundlage für diese Metriken bilden Einschätzungen von Transformationsexperten bei früheren Transformationen.

Der Ansatz ist dabei beliebig konfigurierbar: Als Nutzer kann man sich entweder zwischen sämtlichen Varianten entscheiden und die Bewertung durch die Metriken lediglich als Hilfe sehen, eine gewisse Zahl von unwahrscheinlichen Varianten ausblenden lassen und nur zwischen den wahrscheinlichsten entscheiden, oder alles vollautomatisch durch die Metriken festlegen lassen.

4.2 Erweiterte Spezifikationstechnik für vertikale Modelltransformationen

Die intelligente Propagierung von Änderungen wird kombiniert mit erweiterten Spezifikationstechniken, die es ermöglichen, auch Modelle unterschiedlichen Abstraktionsgrades miteinander zu verknüpfen. Entwickler von Modelltransformationen können so einfacher die Beziehungen zwischen (abstraktem) Systemmodell und (detaillierten) Fachdisziplinen-Modellen beschreiben.

Grundsätzlich wird dabei zwischen *disziplinspezifischen Verfeinerungen* und *disziplinübergreifend relevanten Änderungen* unterschieden. Im Rahmen dieser Arbeit wurde ein Verfahren entwickelt, um die Eigenschaften von Verfeinerungen formal präzise zu definieren. Anhand dieser Definitionen, der sogenannten *Verfeinerungsregeln*, ist die Transformation in der Lage zu erkennen, ob bestimmte Änderungen disziplinübergreifend relevant sind (und daher propagiert werden müssen) oder nicht.

Abb. 4 zeigt ein Beispiel für solch eine Verfeinerung. Auf der rechten Seite (Regelungstechnik) sieht man oben das Zustandsdiagramm, das aus dem Systemmodell links entstan-

den ist. Dieses wurde dann durch Zwischenzustände (fading_N2C und fading_C2N) erweitert (rechts unten). Diese Zwischenzustände sind hier für das reibungslose Umschalten der Regler bei Eintritt und Verlassen eines Konvois zuständig. So eine Änderung ist unter bestimmten Voraussetzungen nicht disziplinübergreifend relevant, nämlich wenn sich dadurch das beobachtbare Verhalten nicht ändert. Dies ist hier der Fall; das Modell rechts unten ist damit immer noch konsistent zum ursprünglichen Systemmodell links.

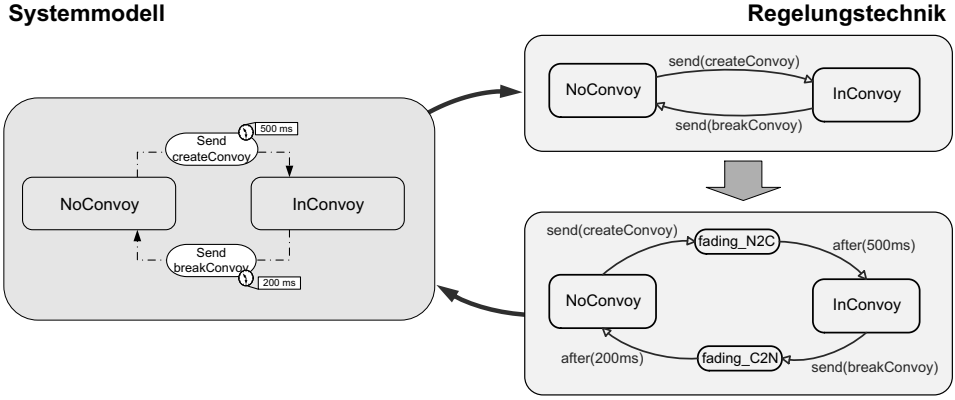

Abb. 4: Verfeinerung eines Zustandsdiagramms in der Regelungstechnik

Zentrale Herausforderung hierbei ist, dass verfeinerte Modellteile trotzdem weiterhin von Änderungen aus anderen Disziplinen betroffen sein können. Hier spielt die oben skizzierte intelligente Propagierung eine zentrale Rolle: Sie bewirkt, dass verfeinerte Modellteile so weit wie möglich erhalten bleiben, sofern die Konsistenzbedingungen dies erlauben.

4.3 Techniken zur interdisziplinären Konfliktlösung

Weiterhin wird die Modelltransformation verknüpft mit bestehenden Lösungen für Modellvergleiche und -konfliktlösungen. So können viele Bearbeitungskonflikte automatisch gelöst werden, ohne den Kern des Transformationsalgorithmus durch direkte Integration von Konfliktlösungsmechanismen unnötig zu verkomplizieren. Für verbleibende Konflikte haben Benutzer zudem spezialisierte Visualisierungs- und Lösungswerkzeuge zur Hand.

4.4 Verbesserte Wartbarkeit von Modelltransformationen

Nachteile von TGGs sind Schwächen in der Verständlichkeit und Wartbarkeit von Transformationsspezifikationen. Zwei Faktoren spielen dabei eine Hauptrolle. Erstens sind TGGs ein deklarativer Formalismus; sie beschreiben also, welche Bedingungen das Ergebnis einer Transformation erfüllen muss, nicht aber wie diese erreicht werden können. Zur Anwendung von TGGs müssen diese daher operationalisiert werden (dies ist die Aufgabe des Transformationsalgorithmus). Für Transformationsentwickler ist es aufgrund dieser „Aufgabenteilung“ schwer mit herkömmlichen Methoden zu erkennen, an welcher Stelle

einer TGG die Ursache eines Fehlers im Ergebnis einer Transformation ist. Zweitens werden TGG-Regeln gerade bei komplexeren Abbildungen schnell sehr groß. Zudem arbeiten sie ausschließlich auf Ebene der abstrakten Syntax der aufeinander abgebildeten Modelle.

Im Rahmen dieser Arbeit wurde daher ein speziell auf TGGs zugeschnittener Debugging-Ansatz entwickelt. Er berücksichtigt einerseits die deklarative Natur von TGGs. Andererseits bietet er Funktionen wie Haltepunkte und Schritt-für-Schritt-Ausführung, die Softwareentwicklern gut bekannt sind. Der Ansatz abstrahiert dabei so weit wie möglich von den Implementierungsdetails des Transformationsalgorithmus. Insbesondere findet das Debugging ausschließlich auf Ebene der TGG-Elemente statt: Haltepunkte werden beispielsweise durch Ereignisse auf TGG-Elementen (wie Knoten, Kanten oder ganzen TGG-Regeln) ausgelöst und nicht durch die Ausführung bestimmter Stellen im Code des Transformationsalgorithmus.

Um die Verständlichkeit der Darstellung von TGG-Regeln zu verbessern, wurde darüber hinaus eine Möglichkeit geschaffen, die konkrete Syntax der aufeinander abgebildeten Modelle direkt in den TGG-Regeln anzuzeigen. Die aus der Arbeit mit den Modellierungssprachen bekannten visuellen Elemente helfen so, sich schnell ein grundlegendes Verständnis einer Regel zu verschaffen.

Literaturverzeichnis

[Be05] Bender, Klaus, Hrsg. Embedded Systems – qualitätsorientierte Entwicklung. Springer, Berlin, Heidelberg, 2005.

[Bu06] Bundesrepublik Deutschland: V-Modell XT 1.4. 2006.

[Fo95] Fohn, Steffen M.; Greef, Arthur; Young, Robert E.; O'Grady, Peter: Concurrent engineering. In (Adelsberger, Heimo H.; Lažanský, Jiří; Mařík, Vladimír, Hrsg.): Information Management in Computer Integrated Manufacturing, Jgg. 973 in Lecture Notes in Computer Science, S. 493–505. Springer Berlin Heidelberg, 1995.

[Ga14] Gausemeier, Jürgen; Korf, Sebastian; Porrmann, Mario; Stahl, Katharina; Sudmann, Oliver; Vaßholz, Mareen: Development of Self-optimizing Systems. In (Gausemeier, Jürgen; Rammig, Franz Josef; Schäfer, Wilhelm, Hrsg.): Design Methodology for Intelligent Technical Systems, Lecture Notes in Mechanical Engineering, S. 65–115. Springer Berlin Heidelberg, 2014.

[GH09] Giese, Holger; Hildebrandt, Stephan: Efficient Model Synchronization of Large-Scale Models. Bericht 28, Hasso Plattner Institute at the University of Potsdam, 2009.

[GW09] Giese, Holger; Wagner, Robert: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling, 8:21–43, 2009.

[Hi12] Hildebrandt, Stephan; Lambers, Leen; Giese, Holger; Petrick, Dominic; Richter, Ingo: Automatic Conformance Testing of Optimized Triple Graph Grammar Implementations. In (Schürr, Andy; Varró, Dániel; Varró, Gergely, Hrsg.): Applications of Graph Transformations with Industrial Relevance, Jgg. 7233 in Lecture Notes in Computer Science, S. 238–253. Springer Berlin Heidelberg, 2012.

[HLR06] Hearnden, D.; Lawley, M.; Raymond, K.: Incremental Model Transformation for the Evolution of Model-Driven Systems. Model Driven Engineering Languages and Systems, 2006.

[In14] What is Systems Engineering? http://www.incose.org/practice/whatissystemseng.aspx.

[La12] Lauder, Marius; Anjorin, Anthony; Varró, Gergely; Schürr, Andy: Bidirectional Model Transformation with Precedence Triple Graph Grammars. In: Proceedings of the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012). Springer Berlin/Heidelberg, 2012.

[La13] Lauder, Marius: Incremental Model Synchronization with Precedence-Driven Triple Graph Grammars. Dissertation, Fachbereich 18 Elektro- und Informationstechnik, Technische Universität Darmstadt, 2013.

[MCT08] Ma, Y.-S.; Chen, G.; Thimm, G.: Paradigm shift: unified and associative feature-based concurrent and collaborative engineering. Journal of Intelligent Manufacturing, 19(6):625–641, 2008.

[MG06] Mens, Tom; Gorp, Pieter Van: A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, 152:125–142, 2006.

[Ri15] Rieke, Jan Dominik: Model Consistency Management for Systems Engineering. Dissertation, Heinz Nixdorf Institut, Universität Paderborn. HNI-Verlagsschriftenreihe, Bd. 335, 2015.

[RS12] Rieke, Jan; Sudmann, Oliver: Specifying Refinement Relations in Vertical Model Transformations. In: Proceedings of the 8th European Conference on Modelling Foundations and Applications (ECMFA 2012). Springer Berlin/Heidelberg, 2012.

[Sc95] Schürr, Andy: Specification of Graph Translators with Triple Graph Grammars. In (Mayr, Ernst W.; Schmidt, Gunther; Tinhofer, Gottfried, Hrsg.): 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'94). Jgg. 903 in Lecture Notes in Computer Science (LNCS), Springer Verlag, Heidelberg, S. 151–163, 1995.

[St73] Stachowiak, Herbert: Allgemeine Modelltheorie. Springer-Verlag, Wien, New York, 1973.

[Ve04] Verein Deutscher Ingenieure: Design Methodology for Mechatronic Systems. Beuth Verlag GmbH, 2004.

[Xi09] Xiong, Yingfei; Song, Hui; Hu, Zhenjiang; Takeichi, Masato: Supporting Parallel Updates with Bidirectional Model Transformations. In: Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations (ICMT '09). Springer-Verlag, 2009.

Jan Dominik Rieke, geboren am 7. März 1982, studierte Informatik mit Nebenfach Physik an der Universität Paderborn. Von 2009 bis 2014 arbeitete er als wissenschaftlicher Mitarbeiter und Promotionsstudent der International Graduate School „Dynamic Intelligent Systems“ in der Fachgruppe „Softwaretechnik“ bei Prof. Dr. Wilhelm Schäfer. In dieser Zeit hat er in verschiedenen Forschungsprojekten wie dem Sonderforschungsbereich 614 „Selbstoptimierende Systeme des Maschinenbaus“ mitgearbeitet. Von 2012 bis 2014 hat er darüber hinaus die Projektkoordination des Forschungsprojekts „ENTIME – Entwurfstechnik Intelligente Mechatronik“ übernommen.