Formale Spezifikationsebene*

Mathias Soeken

AG Rechnerarchitektur
Universität Bremen
msoeken@informatik.uni-bremen.de

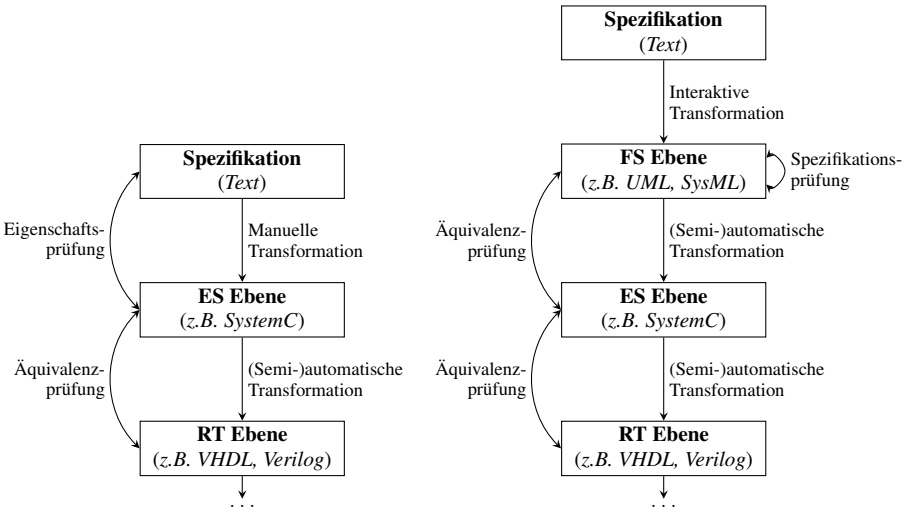
Abstract: Während die hardwarenahen Abstraktionsebenen im Entwurf von komplexen sicherheitskritischen Systemen mittlerweile gut verstanden sind, stellt insbesondere die korrekte Transformation auf höheren Ebenen (wie beispielsweise die Überführung einer textuellen Spezifikation in eine Implementierung) eine große Herausforderung da. Dies lässt sich insbesondere mit der großen konzeptuellen Diskrepanz dieser beiden Ebene erklären. Als Lösungsvorschlag wurde in der hier zusammengefassten Dissertation eine neue Abstraktionsebene zwischen Spezifikation und Implementierung eingeführt, die (a) eine semiautomatische Übersetzung der informellen Spezifikation in ein formales Modell erlaubt und (b) Verifikationsmethoden auf dem resultierenden Modell ermöglicht. Dadurch wird nicht nur die Qualität des Entwurfs verbessert; auch Fehler lassen sich so früh im Entwurf entdecken, was zu Zeitersparnissen führt.

1 Einführung

Der Entwurf von eingebetteten Systemen, die heutzutage aus bis zu Milliarden von Einzelkomponenten bestehen, ist eine der komplexesten Aufgaben mit denen sich Wissenschaftler und Ingenieure auseinandersetzen. Während es vor 40 Jahren noch möglich war solche Systeme Gatter für Gatter auf dem Reißbrett zu entwerfen, ist dieses Vorgehen heute praktisch wegen der stetig steigenden Komplexität nicht mehr umsetzbar. Um den Entwurf in den Griff zu bekommen, haben Wissenschaftler innerhalb der letzten Jahrzehnte einen ausgeklügelten und gründlich durchdachten Entwurfsablauf entwickelt, welcher aus einer Vielzahl von unterschiedlichen Abstraktionsebenen besteht, die das zu entwerfende System sukzessive verfeinern. Dieser Ablauf beginnt bei einer überwiegend textuellen Spezifikationen und mündet schließlich in einer ausgeflachten Transistornetzliste mit präzisen Layoutinformationen.

Der Entwurfsablauf wie er heutzutage üblicherweise Anwendung findet ist dargestellt in Abbildung 1(a). Dessen Startpunkt und somit auch die abstrakteste Beschreibung ist eine informelle natürlichsprachliche Spezifikation. Um jedoch auch nur die einfachste automatische Methode anwenden zu können, muss eine formale Beschreibung vorliegen, die maschinenlesbar ist. Aus diesem Grunde wird ausgehend von der textuellen Spezifikation

*Englischer Titel der Dissertation: “Formal Specification Level”. Die Arbeit wurde betreut von Prof. Dr. Rolf Drechsler, Leiter der Arbeitsgruppe Rechnerarchitektur an der Universität Bremen.



(a) Traditioneller Entwurfsablauf (b) Neuer Entwurfsablauf

Abbildung 1: Gegenüberstellung der Entwurfsabläufe

eine initiale Implementierung in der elektronischen Systemebene (engl.: *Electronic System Level*, kurz ESL, [BM09]) generiert. Auf dieser Ebene stehen softwarenahe Programmiersprachen wie SystemC zur Verfügung. SystemC ist eine C++-Klassenbibliothek für ereignisgesteuerte Simulation nebenläufiger Prozesse. Die Systemebene erlaubt die Ausführung und Simulation des zu entwerfenden Systems, benötigt jedoch keine konkreten Details über die Umsetzung als Hardware. Ausgehend von dieser Beschreibung wird das System sukzessive verfeinert, was zu immer detailreicheren Beschreibungen auf Register-Transferebene (engl.: *Register Transfer Level*, kurz RTL), Gatterebene und physikalischer Ebene führt. Am Ende dieses Prozesses kann das resultierende System zur Fabrikation gesendet werden.

Da eingebettete Systeme oft in sicherheitskritischen Bereichen wie Avionik-, Automobil- und Medizinanwendungen zum Einsatz kommen, ist insbesondere ein *korrekter* Entwurf von höchster Bedeutung. Um die Korrektheit sicherzustellen, werden üblicherweise die Transformationen von einer Abstraktionsebene auf die nächst präzisere auf Äquivalenz verglichen. Da jedoch kein formales Modell für die Spezifikation vorliegt, kann die textuelle Spezifikation nicht automatisch mit dem Modell auf der Systemebene verglichen werden. Die Lücke zwischen diesen beiden Ebenen ist die größte im Entwurfsablauf. Dies wird insbesondere dadurch ersichtlich, dass die Spezifikation *informell* beschreibt *was* entworfen werden soll, die Systemebene hingegen *formal* beschreibt *wie* das System entworfen werden soll. Da zusätzlich zu dieser konzeptuellen Diskrepanz die Spezifikation manuell in ein Modell der Systemebene übersetzt werden muss, ist dieser erste Schritt im Entwurf besonders anfällig für Fehler.

Zur Überprüfung, ob die Intentionen aus der Spezifikation auf Systemebene umgesetzt werden, wird heutzutage Eigenschaftsprüfung als Alternative zum Äquivalenzvergleich

eingesetzt. Dazu werden formale temporallogische Eigenschaften aus der Spezifikation extrahiert welche mithilfe von *Model Checking* [CGP99] mit dem Modell auf Systemebene abgeglichen werden. Darauf aufsetzend existieren Techniken, welche die Vollständigkeit der formalen Abdeckung sicherstellen [GD10]. Das größte Hindernis bleibt jedoch weiterhin bestehen: die Spezifikation ist in natürlicher Sprache gegeben und eine formale Beschreibung muss manuell von ihr abgeleitet werden. Motiviert durch dieses Problem haben Wissenschaftler kürzlich damit begonnen, Techniken zu entwickeln, welche die Lücke zwischen Spezifikations- und Systemeebene schließen [Har12].

Aufbauend auf diesen Arbeiten wird in der Dissertation die formale Spezifikationsebene (engl.: *Formal Specification Level*, kurz: FSL, [DSW12a][1]) als neue Abstraktionsebene eingeführt wie es in Abbildung 1(b) illustriert ist. Die Beschreibungsmittel in dieser Ebene basieren auf Modellierungssprachen mit denen *formal* beschrieben wird *was* entworfen worden soll. Somit verbindet die FSL die textuelle Spezifikationsebene mit der ESL in idealer Weise. Modellierungssprachen wie die *Unified Modeling Language* (UML, [RJB99]), die *System Modeling Language* (SysML, [Wei07]) kombiniert mit Ausdrücken der *Object Constraint Language* (OCL, [WK99]) erlauben eine genaue Beschreibung der Syntax und Semantik für die Modelle auf der FSL. Durch ihre Abstraktheit erlauben diese Modelle eine geeignete Beschreibung der Spezifikationen und ermöglichen zusätzlich die Anwendung automatischer Analysen durch ihre formale Natur. Mit Einführung dieser neuen Abstraktionsebene liefert die Dissertation folgende Beiträge:

- Da die FSL näher zur textuellen Spezifikation ist — auf beiden Ebenen wird beschrieben, was entworfen werden soll — wird eine semiautomatische Übersetzung von informellen Spezifikationselementen in formale Modelle ermöglicht. Techniken aus den Bereichen der maschinellen Textverarbeitung [JM00], Informationsextraktion [CL96] und Wissensrepräsentation [Mil95] werden zu diesem Zweck verwendet. Schon mit einfachen grammatikalische Analysen können beispielsweise die Komponenten des Systems (repräsentiert durch Nomen), ihre Funktionalität (repräsentiert durch Verben) sowie ihre Eigenschaften (repräsentiert durch Adjektive) extrahiert werden. Anstatt auf einen automatischen Ansatz anzustreben, werden in der Dissertation interaktive Methoden entwickelt, mit denen der Entwickler zusammen mit dem Computer die informelle Spezifikation in formale Modelle übersetzt. Dadurch wird die Qualität der Transformation signifikant verbessert.

- Durch neue Verifikationsmethoden auf Modellebene können kritische Fehler in der Spezifikation früher entdeckt werden; sogar noch bevor eine Implementierung geschrieben ist. Diese Fehler sind meist von konzeptueller Art und eine späte Entdeckung impliziert oft aufwändige Korrekturen, die sich über viele Ebenen ausstrecken. Die Verifikationsmethoden sind bezüglich Skalierbarkeit optimiert und betrachten sowohl strukturelle als auch verhaltensbasierte Eigenschaften des Modells.

In dieser Zusammenfassung soll insbesondere auf den zweiten Punkt eingegangen werden. Zunächst wird im nächsten Abschnitt ein detaillierter Blick auf die neue Abstraktionsebene geworfen.

[1] Die Verfahren und Ergebnisse aus der Dissertation werden derzeit für ein Buch aufgearbeitet [SD14].

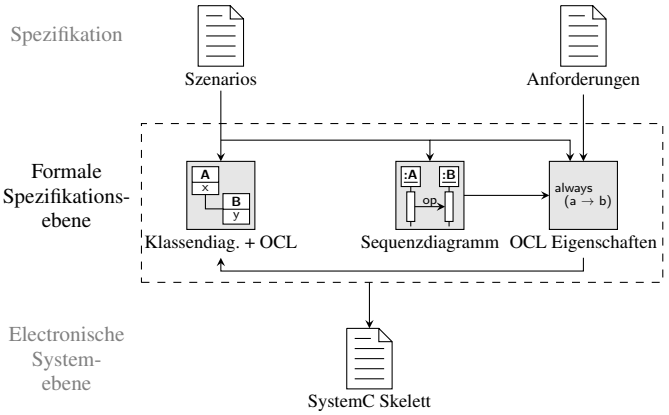


Abbildung 2: Überblick über die formale Spezifikationsebene

2 Die Formale Spezifikationsebene

Abbildung 2 zeigt einen detaillierten Überblick über die vorgeschlagenen Erweiterungen des vorgegenwärtigen Entwurfablaufes. Die Hauptziele sind eine semiautomatische Übersetzung der textuellen Spezifikation in eine Beschreibung auf der elektronischen Systemebene (z.B. SystemC) sowie die Möglichkeit zur Anwendung von Methoden für die Korrektheitsprüfung des formalen Modells selbst. Wir unterscheiden auf der textuellen Spezifikationsebene zwischen Szenarien und Anforderungen als Beschreibungsmittel. Anforderungen können sowohl funktionale als auch nicht-funktionale Eigenschaften beschreiben. In einer Automobilanwendungen können beispielsweise “Die Fenster des Autos können automatisch bedient werden.” und “Der Airbag löst innerhalb von höchstens 30 ms aus.” Anforderungen sein. Anforderungen werden verwendet, um formale Eigenschaften aus ihnen zu generieren, die automatisch auf dem Modell der Systemebene geprüft werden können. Für eine Testfallgeneration bieten sich Szenarien an, da diese konkrete Anwendungsfälle beschreiben. Ein Beispiel für ein Szenario ist “Nachdem die Zündung geschaltet worden ist, startet der Motor.” Szenarien können einfach in Testfälle übersetzt werden, sind jedoch nicht zur vollständigen Verifikation geeignet, da nicht alle möglichen Fälle berücksichtigt werden.

Mit der FSL können bei gegebenen Szenarien und Anforderungen in natürlicher Sprache eine initiale SystemC-Implementierung semiautomatisch generiert werden. Zusätzlich kann die Spezifikation auf konzeptuelle Korrektheit automatisch geprüft werden. Dazu werden innerhalb der FSL zwei Schritte vollzogen, die im Folgenden erläutert werden.

Im ersten Schritt werden Szenarien und Anforderungen in formale Modelle übersetzt. Techniken der maschinellen Textverarbeitung werden angewendet, um die folgenden Teilaufgaben dieses ersten Schrittes zu lösen:

- **Extraktion der Struktur:** Mithilfe der grammatikalischen Analyse von Sätzen innerhalb der textuellen Spezifikation können Basiskomponenten des Systems extra-

hiert werden. Aus den extrahierten Informationen können beispielsweise Klassendiagramme erstellt werden, die eine erste formale Beschreibung der Struktur bieten.

- **Extraktion des Verhaltens:** Auf ähnliche Weise lassen sich auch Ausführungssequenzen extrahieren und in Form von Sequenzdiagrammen darstellen.
- **Extraktion von formalen Eigenschaften:** Nachdem die Struktur extrahiert worden ist, können basierend auf dem formalen Modell auch natürlichsprachliche Anforderungen in formale Eigenschaften, beispielsweise in Form von OCL Ausdrücken, extrahiert werden.

Als Resultat liefert der erste Schritt eine formale Beschreibung des Systems in Form von Modellen. Konkret sind es in diesem Fall Klassen- und Sequenzdiagramme, die um OCL Ausdrücke angereichert sind.

Im zweiten Schritt werden diese formalen Beschreibungen verwendet, um initiale automatische Analysen auszuführen, welche die Spezifikation auf eine konzeptuelle Korrektheit überprüfen:

- **Verifikation struktureller Aspekte.** Bei großen Spezifikationen ist es nicht unwahrscheinlich, dass widersprüchliche Anforderungen enthalten sind. Dies bedeutet, dass es keine gültige Implementierung der Spezifikation geben kann. Bei zu vielen Anforderungen und großen Modellen kann diese Aufgabe nicht mehr manuell bewältigt werden.
- **Verifikation des Verhaltens.** Wenn zusätzlich zu den Anforderungen auch die Operationen des Modells, welche in Form von Vor- und Nachbedingungen spezifiziert sind, betrachtet werden, lassen sich Erreichbarkeitsanalysen automatisch durchführen. Dies beinhaltet beispielsweise Verfahren, ob ein nichterlaubter Zustand erreicht werden kann oder ob es möglich ist in einen Zustand zu gelangen, der nicht mehr verlassen werden kann.

Da diese Verifikationstechniken bereits angewendet werden können, bevor eine konkrete Implementierung des Systems vorliegt, können kritische Fehler in der Spezifikation bereits in einer sehr frühen Phase des Entwurfs entdeckt und behoben werden.

3 Abbildung von Szenarien in die FSL

Zur semiautomatischen Extraktion von Informationen aus der informellen Spezifikation mithilfe von maschineller Textverarbeitung wurden in der Dissertation verschiedene Ansätze vorgestellt, die in diesem Abschnitt kurz zusammengefasst sind. In [SWD12a] ist ein Ansatz vorgestellt worden, der den Entwickler bei der Übersetzung von natürlichsprachlichen Szenarien in formale Modelle unterstützt. Mittels syntaktischer Analyse werden dabei Klassen, Attribute, Operationen und Assoziation extrahiert. Dabei wird die natürliche Sprache nicht eingeschränkt. Mögliche Mehrdeutigkeiten werden vom Computer erkannt

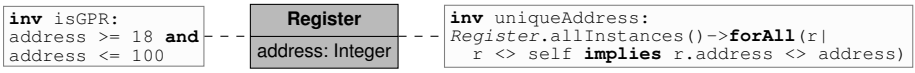


Abbildung 3: Klassendiagramm

und in einem Dialog mit dem Entwickler aufgelöst [DSW12b]. Dies führt insbesondere zu einer klareren Spezifikation, denn Texte, die durch einen Computer missverstanden werden können, werden wahrscheinlich auch von anderen Entwicklern falsch interpretiert. Das Extraktionsverfahren wurde auch in Form einer integrierten Entwicklungsumgebung implementiert [KSKD13].

4 Verifikation von formalen Modellen

Durch den ersten Schritt in der FSL wurde ein formales Modell aus der Spezifikation extrahiert. Im zweiten Schritt soll das Modell auf konzeptuelle Konsistenz überprüft werden, da es bei vielen Anforderungen möglich ist, dass sie widersprüchlich sind. Zu diesem Zweck sind in der Dissertation verschiedene Ansätze entwickelt worden [SWK+10, SWD11b, SWD11a, WSD12], von denen in diesem Abschnitt einige vorgestellt werden.

4.1 Verifikation struktureller Aspekte

Falls die formale Repräsentation des Designs Widersprüche enthält, kann keine valide Implementierung existieren, die der Spezifikation genügt. Verifikationsmethoden auf der FSL erlauben diese Fehler zu erkennen, noch bevor eine Zeile Quellcode geschrieben worden ist.

Strukturelle Aspekte werden insbesondere durch Invarianten in OCL beschrieben, die mögliche Instanzen eines Klassendiagramms einschränken. Zu restriktive Invarianten können zu einer Inkonsistenz des Modells führen.

Beispiel 1. *Abbildung 3 zeigt ein einfaches Klassendiagramm, das ein Register mit einer Adresse modelliert. Zusätzliche Invarianten fordern, dass in einem Objektdiagramm jedes Register eine Adresse zwischen 18 und 100 hat und dass je zwei Register paarweise unterschiedliche Adressen haben. Das Klassendiagramm ist genau dann konsistent, wenn es höchstens 83 Registerinstanzen im Objektdiagramm gibt, da ansonsten nicht ausreichend Adressen zugewiesen werden können.*

Verifikationsmethoden versuchen die Konsistenz eines Klassendiagramms durch das Auffinden einer validen Instanz zu bestätigen. Der in der Dissertation entwickelte Ansatz *ocl2smt* [SWK+10] ist insbesondere in Hinsicht auf Skalierbarkeit optimiert worden. Dazu wird das Konsistenzproblem als Instanz des Erfüllbarkeitsproblems (SAT) formuliert und mit entsprechenden Beweisern automatisch gelöst [BHvMW09]. Im Folgenden wird ein Experiment erläutert, dass die signifikante Performanzsteigerung durch die Transfor-

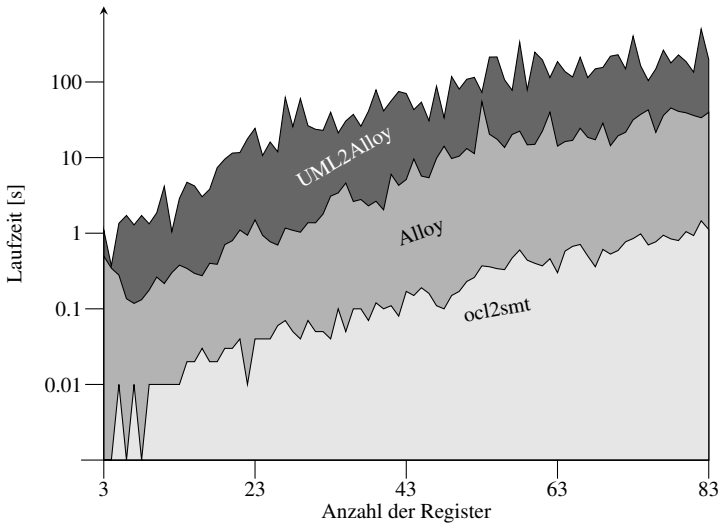


Abbildung 4: Ergebnis des Experiments zum Performanzvergleich

mation in eine SAT Instanz illustriert.

Ein naiver Ansatz wie beispielsweise in [GKH09] illustriert löst das Problem mit ausschöpfender Suche. Der Suchraum für dieses Problem ist allein durch die Anzahl der Instanzen im Objektdiagramm und dem einzigen Attribute *address* charakterisiert. Bei drei Registern benötigt dieser enumerative Ansatz 2,952 Sekunden, bei vier Registern 112 Sekunden. Da alle betrachteten Objektdiagramme in geordneter Reihenfolge validiert werden, lässt sich eine Simulationsgeschwindigkeit von ca. 150.000 Objektdiagrammen pro Sekunde abschätzen. Dadurch lässt sich extrapolieren, dass bereits für ein Objektdiagramm mit sieben Registern mehr als 3,5 Jahre benötigt werden, um ein valides zu finden.

Es gibt andere Ansätze wie z.B. *Alloy* [Jac06], die das Problem auch mithilfe einer Transformation in eine SAT Instanz lösen, jedoch nicht Klassendiagramme mit OCL Invarianten als Eingabesprache verwenden. Abhilfe schafft hier der Ansatz *UML2Alloy* [ABGR07], der Klassendiagramme mit OCL Ausdrücken in ein für Alloy verständliches Format überführt. Im Experiment wurden *Alloy*, *UML2Alloy* und der in der Dissertation entwickelte Ansatz *ocl2smt* verwendet, um für das Klassendiagramm in Abbildung 3 konsistente Objektdiagramme zu finden. Dazu wurde die Anzahl der Register im Objektdiagramm bis zu einer Maximalanzahl von 83 sukzessive erhöht. Zudem wurde als SAT Beweiser MiniSAT [ES03] verwendet, da dieser auch in *Alloy* zum Einsatz kommt.

Die Ergebnisse des Experiments sind in Abbildung 4 illustriert. Alle Ansätze finden Objektdiagramme in weniger als 10 Minuten (die Laufzeit ist auf einer logarithmischen Skala aufgetragen). *UML2Alloy* benötigt am meisten Zeit. *Alloy* ist signifikant schneller, wenn das Problem direkt in der Sprache von *Alloy* formuliert und nicht automatisch übersetzt wird. Der Unterschied in der Laufzeit wächst exponentiell mit Anzahl der Register. Diese Performanzsteigerung erhält man ein weiteres Mal, wenn man das Problem mit *ocl2smt* direkt in eine SAT Instanz formuliert. Um eine möglichst gute Laufzeit zu erhalten, müs-

sen also Transformationsschritte vermieden werden. Auch wenn hier nur ein Beispiel von kleiner Größe betrachtet worden ist, ist der Einfluss der verwendeten Verifikationsmethode dennoch signifikant. Erstens ist der Effekt viel stärker bei größeren Modellen und zweitens kann es sich bei dem Beispiel um ein Teilmodell eines komplexeren Modells handeln.

Die Skalierbarkeit von *ocl2smt* erlaubte unter anderem die Anwendbarkeit größerer Instanzen bei der Testfallgenerierung für Modelltransformationen [GS13]. Außerdem wurde in [WSD12] der erste Ansatz vorgestellt, der Ursachen für inkonsistente Klassendiagramme ermittelt.

4.2 Verifikation des Verhaltens

Auch ein konzeptuelles Verhalten kann bereits auf Basis der Modelle auf der FSL modelliert werden. Dazu werden Vor- und Nachbedingungen verwendet, die an Operationen annotiert werden, ohne eine explizite Implementierung zu fordern. Eine Vorbedingung beschreibt in welchen Zuständen eine Operation aufgerufen werden kann und eine Nachbedingung beschreibt wie sich der Zustand durch einen Operationsaufruf ändert.

Auf Basis dieser durch Vor- und Nachbedingungen annotierten Modelle wurde in der Dissertation erstmals ein Verfahren entwickelt [SWD11b], dass durch *Bounded Model Checking* (BMC, [BCC$^+$03]) inspiriert ist. Durch die Vor- und Nachbedingungen kann eine Ausführungssemantik modelliert werden, die sich auch in einer SAT Instanz formulieren lässt. Dadurch lassen sich dann Verifikationsaufgaben formulieren wie beispielsweise, ob alle Operationen ausgeführt werden können oder ob es einen Deadlock gibt, d.h. ein Zustand, in dem keine Operation aufgerufen werden kann. Da auf Modellebene keine Implementierung der Operationen vorliegt, lässt sich eine viel höhere Skalierbarkeit im Vergleich zur Verifikation auf der ESL erreichen. Das Verfahren wurde auch angewendet um globale OCL Invarianten semiautomatisch in lokale Vor- und Nachbedingungen zu transformieren [SWD12b].

5 Zusammenfassung

In der Dissertation ist eine neue Abstraktionsebene — die formale Spezifikationsebene (FSL) — für den Entwurf von komplexen Systemen vorgeschlagen und detailliert ausgearbeitet worden, welche die informelle und vorwiegend textuelle Spezifikation mit der initialen Implementierung auf der Systemebene verbindet. Während die Spezifikation informell beschreibt, was entworfen werden soll und die Implementierung formal beschreibt, wie es entworfen werden soll, bietet die FSL hier eine ideale Verbindung, da sie formal beschreibt was entworfen werden soll. Somit ist es erstmals möglich Verifikationsmethoden bereits vor der Implementierungsphase automatisch auszuführen, was zu erheblichen Zeiteinsparungen führt. Außerdem wurden Verfahren vorgestellt, die eine interaktive und qualitätsorientierte Übersetzung der informellen Spezifikation in ein formales Model auf Basis von maschineller Textverarbeitung ermöglichen. Dies trägt zusätzlich zur Qualität des zu entwerfenden Systems bei. Ausgewählte Verfahren der Dissertation wurden in die-

ser Zusammenfassung kurz vorgestellt.

Danksagung

Die Arbeit wurde unterstützt durch die DFG im Rahmen des Reinhart Koselleck Projektes "Entwicklung eines durchgängigen Verifikationsablaufes für den ESL Entwurf" (DR 287/23-1) und durch das BMBF im Projekt "SPECifIC – Qualitätsgetriebener Entwurf durch Formale Spezifikation und Funktionales Änderungsmanagement" (01IW13001).

Literatur

[ABGR07] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg und Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. In *Int'l Conference on Model Driven Engineering Languages and Systems*, Seiten 436–450, 2007.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman und Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren und Toby Walsh, Hrsg. *Handbook of Satisfiability*, Jgg. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BM09] Brian Bailey und Grant Martin. *ESL Models and Their Application: Electronic System Level Design and Verification in Practice*. Springer, 2009.

[CGP99] Edmund M. Clarke, Jr., Orna Grumberg und Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CL96] James R. Cowie und Wendy G. Lehnert. Information Extraction. *Commun. ACM*, 39(1):80–91, 1996.

[DSW12a] Rolf Drechsler, Mathias Soeken und Robert Wille. Formal Specification Level: Towards verification-driven design based on natural language processing. In *Forum on Specification and Design Languages*, Seiten 53–58, 2012.

[DSW12b] Rolf Drechsler, Mathias Soeken und Robert Wille. Towards dialog systems for assisted natural language processing in the design of embedded systems. In *Int'l Design & Test Symposium*, 2012.

[ES03] Niklas Eén und Niklas Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, Seiten 502–518, 2003.

[GD10] Daniel Große und Rolf Drechsler. *Quality-Driven SystemC Design*. Springer, 2010.

[GKH09] Martin Gogolla, Mirco Kuhlmann und Lars Hamann. Consistency, Independence and Consequences in UML and OCL Models. In *Tests and Proofs*, Seiten 90–104, 2009.

[GS13] Esther Guerra und Mathias Soeken. Specification-Driven Model Transformation Testing. *Software and System Modeling*, 2013. accepted.

[Har12] Ian G. Harris. Extracting design information from natural language specifications. In *Design Automation Conference*, Seiten 1256–1257, 2012.

[Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[JM00] Daniel Jurafsky und James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2000.

[KSKD13] Oliver Keszocze, Mathias Soeken, Eugen Kuksa und Rolf Drechsler. lips: An IDE for Model Driven Engineering Based on Natural Language Processing. In *Workshop on Natural Language Analysis in Software Engineering*, Seiten 31–38, 2013.

[Mil95] George A. Miller. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11):39–41, 1995.

[RJB99] James E. Rumbaugh, Ivar Jacobson und Grady Booch. *The unified modeling language reference manual*. Addison-Wesley-Longman, 1999.

[SD14] Mathias Soeken und Rolf Drechsler. *Formal Specification Level*. Springer, 2014. in preparation.

[SWD11a] Mathias Soeken, Robert Wille und Rolf Drechsler. Encoding OCL Data Types for SAT-Based Verification of UML/OCL Models. In *Tests and Proofs*, Seiten 152–170, 2011.

[SWD11b] Mathias Soeken, Robert Wille und Rolf Drechsler. Verifying Dynamic Aspects of UML Models. In *Design, Automation and Test in Europe*, Seiten 1077–1082, 2011.

[SWD12a] Mathias Soeken, Robert Wille und Rolf Drechsler. Assisted Behavior Driven Development Using Natural Language Processing. In *Int'l. Conference on Objects, Models, Components, Patterns*, Seiten 269–287, 2012.

[SWD12b] Mathias Soeken, Robert Wille und Rolf Drechsler. Eliminating Invariants in UML/OCL Models. In *Design, Automation and Test in Europe*, Seiten 1142–1145, 2012.

[SWK+10] Mathias Soeken, Robert Wille, Mirco Kuhlmann, Martin Gogolla und Rolf Drechsler. Verifying UML/OCL models using Boolean satisfiability. In *Design, Automation and Test in Europe*, Seiten 1341–1344, 2010.

[Wei07] Tim Weilkiens. *Systems engineering with SysML / UML - modeling, analysis, design*. Morgan Kaufmann, 2007.

[WK99] Jos Warmer und Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Longman, 1999.

[WSD12] Robert Wille, Mathias Soeken und Rolf Drechsler. Debugging of Inconsistent UML/OCL Models. In *Design, Automation and Test in Europe*, Seiten 1078–1083, 2012.

Mathias Soeken ist PostDoc an der Universität Bremen in der Arbeitsgruppe Rechnerarchitektur und Wissenschaftler bei der DFKI GmbH in der Gruppe Cyber-Physical Systems. Seine Forschungsinteressen sind formale Verifikation, maschinelle Textverarbeitung und zukünftige Computertechnologien.