Scaling-Up von V&V-Techniken durch Integration und Abstraktion

S. Helke, A. Nordwig, T. Santen, D. Sokenou

Technische Universität Berlin
Institut für Kommunikations- und Softwaretechnik
Sekr. FR 5-6, Franklinstr. 28/29, D-10587 Berlin
email:{helke|nordwig|santen|dsokenou}@cs.tu-berlin.de

Zusammenfassung Wir beschreiben einen Ansatz, die Komplexität von Verifikationsaufgaben durch Abstraktion von irrelevanten Details eines Modells und durch die Integration verschiedener Verifikationstechniken zu reduzieren, um diese so für Beschreibungen software-intensiver Systeme anwendbar zu machen. Wir gehen insbesondere auf die Wechselwirkung zwischen Verifikations- und Testaufgaben in der Qualitätssicherung ein, betrachten Abstraktionstechniken, die den Anwendungsbereich von automatischem Beweisen und Modelchecking vergrößern, und wir gehen auf die spezielle Problematik der Verifikation kontinuierlicher Systeme ein.

1 Einleitung

Die Entwicklung eines software-intensiven Systems erfordert den parallelen Einsatz unterschiedlicher Beschreibungs- und Verifikationstechniken, um die verschiedenen Aspekte eines solchen Systems adäquat zu erfassen und ihre Realisierung in Entwurf und Code zu überprüfen. Eine Reihe spezifischer Probleme machen die Verifikation und Validation (V&V) solcher Systeme besonders schwierig:

Größe Die schiere Größe des Systems macht es sehr schwierig, eine Beschreibung des Gesamtsystems in hohem Detaillierungsgrad zu überschauen. Das gilt sowohl für Menschen, als auch für automatische Verfahren, die in der Praxis oft genug nicht an der Komplexität, sondern am Detaillierungsgrad der Beschreibungen scheitern. Andererseits ist Automatisierung von V&V-Aufgaben gerade für große Systeme unerlässlich.

Aspekte Das System lässt sich nicht auf einen „Hauptaspekt“ reduzieren, wie z.B. das reaktive Verhalten, der alle relevanten Eigenschaften abdeckt. Es ist notwendig, verschiedene Beschreibungs- (und evtl. auch Implementierungs-) paradigmen einzusetzen, um die Software adäquat zu beschreiben und zu realisieren.

Verifikationsprozess Es ist in der Praxis unmöglich, das Gesamtsystem (in welchem Sinne auch immer) vollständig zu verifizieren. Die Komplexität des Systems lässt weder eine vollständige formale Verifikation, noch einen Systemtest mit hohem Überdeckungsgrad zu.

Hybridität Verifikationstechniken für Software nutzen üblicherweise den diskreten Charakter von Software aus und sind nicht auf kontinuierliche Systeme ausgerichtet. Zwar arbeitet auch die Software zur Verarbeitung kontinuierlicher Größen

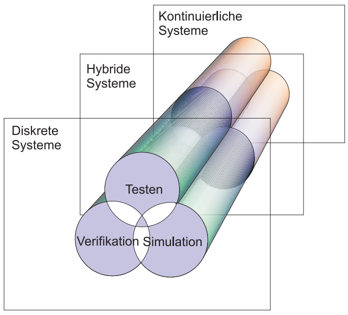

Abbildung 1. Verzahnung von V&V-Techniken für verschiedene Systemklassen.

intern mit diskreten Zuständen, trotzdem erfordern kontinuierliche Umgebungsgrößen eine andere Herangehensweise zur Verifikation und Validation.

Im Folgenden diskutiern wir einen Ansatz, diese Probleme durch die Kombination verschiedener Techniken anzugehen. Wir bieten keine technisch ausformulierten Lösungen an, sondern formulieren Forschungsziele, um V&V-Aufgaben für software-intensive Systeme in Zukunft besser bewältigen zu können.

Grundsätzlich wollen wir den Anwendungsbereich von V&V-Techniken dadurch vergrößern, dass wir systematisch Synergien zwischen verschiedenen Techniken ausnutzen und Doppelarbeit vermeiden, diese Techniken also *integrieren*. Darüber hinaus sind *Abstraktionsansätze* notwendig, die die Detailfülle in Beschreibungen großer Systeme soweit reduzieren, dass bekannte Techniken anwendbar werden.

Diese Herangehensweise hat Ähnlichkeiten zu den Bemühungen um die Integration unterschiedlicher Beschreibungstechniken, die in den letzten Jahren verstärkt angegangen wird [13, 4, 2]. Im Gegensatz zu der Integration von Beschreibungen, die unterschiedliche Aspekte eines Systems modellieren, ist es jedoch sehr viel schwieriger, Verifikations- und Validationsaufgaben orthogonal zu formulieren und Techniken zu ihrer Durchführung verzahnt einzusetzen.

Im Rest des Beitrags untersuchen wir formale Verifikations-, Test- und Simulationstechniken in bezug auf ihr Potential zur wechselseitigen Ergänzung. Abschließend skizzieren wir eine Vision zur gemeinsamen Anwendung dieser Techniken im Bereich software-intensiver Systeme.

2 Modelchecking – Ausgangspunkt für V&V Integrationen

Die systematische Entwicklung von Verifikationstechniken wurde in den letzten Jahren durch intensive Forschung vorangetrieben. In industriellen Anwendungen kommen bereits Verifikationswerkzeuge zum Einsatz, um höhere Sicherheitsstandards für Software zu erzielen. Besonders erfolgreich werden Model Checker im Hardwarebereich eingesetzt, wo auch bei größeren Systemen die Komplexität noch beherrscht werden kann. Aber auch die Entwicklung eingebetteter sicherheitskritischer Softwaresysteme kann durch geeignete Analysewerkzeuge unterstützt werden. Hauptanwendungsbereich bleiben jedoch, aufgrund ihrer geringeren Komplexität, *diskrete Systeme*.

Modelchecking ist eine vollautomatische Verifikationstechnik, in der ein endlicher Zustandsraum vollständig (symbolisch oder explizit) aufgezählt wird, um die Gültigkeit einer temporallogischen Eigenschaft nachzuweisen. Es gibt intensive Bestrebungen, Modelchecking mit anderen V&V-Techniken zu kombinieren. Im Folgenden wollen wir drei Varianten kurz vorstellen.

Theorembeweisen Praktische Anwendungen basieren häufig auf unendlichen oder zu großen Zustandsräumen, so dass Eigenschaften nur auf einer geeigneten endlichen Abstraktion des realen Modells verifiziert werden können. Im Gegensatz zu Model Checkern können in Theorembeweisern auch unendliche Zustandsräume erfasst werden, so dass hier der Nachweis der Korrektheit einer Abstraktion erbracht werden kann. Eine Abstraktion ist bezüglich einer zu prüfenden Eigenschaft P dann korrekt, wenn lediglich von Details abstrahiert wird, die für P nicht relevant sind [17].

Testen Das Vorgehen zur Abstraktion von unendlichen Zustandssystemen beim Modelchecking ist ähnlich zur Klassifizierung von Testdaten im Äquivalenzklassentest. Ein abstrakter Zustand repräsentiert die Eigenschaften von mehreren – möglicher Weise unendlich vielen – konkreten Zuständen. Analog werden in einer Testklasse Daten mit gemeinsamen Eigenschaften zusammengefasst. Vor diesem Hintergrund ist es sinnvoll, Testdaten aus dem abstrakten Modell abzuleiten [8]. Dieser Ansatz wird bei nichtsymbolischen Model Checkern verfolgt, da hier die Algorithmen bereits darauf basieren, Systemläufe – also Testdaten – explizit zu berechnen, die Grundlage für eine Testdurchführung sein können.

Simulation Das Verletzen einer temporallogischen Eigenschaft des zu überprüfenden Modells wird beim Modelchecking durch die Rückgabe eines Gegenbeispiels, i.a. ein *Systemlauf*, angezeigt. Diesen Lauf zu simulieren ist hilfreich für das Verständnis des aufgedeckten Fehlers und damit eine sinnvolle Vorarbeit für einen Neuentwurf des Modells.

Im Folgenden wollen wir auf die gewinnbringende Kombination von Modelchecking und Theorembeweisen zur Abstraktion von unendlichen Zustandssystemen etwas genauer eingehen.

Die *Datenabstraktion* ist ein spezielles Verfahren, in dem für Variablen mit unendlichen Datendomains über charakteristische Prädikate ein abstraktes System in einem Theorembeweiser automatisch konstruiert werden kann [21]. Diese Technik wollen wir an einem Beispiel etwas genauer vorstellen. Der obere Teil der Abb. 2 zeigt eine Statechartspezifikation, in der der gegenseitige Ausschluss zweier konkurrierender Prozesse mithilfe einer Semaphore S modelliert wurde. Die Prozesse sind durch die Variablen $PC1$ und $PC2$ repräsentiert, die einen der Zustände $idle$, $critical$ oder $waiting$ annehmen können. Das Vorhaben eines Prozesses, einen kritischen Abschnitt zu betreten, wird durch die Ereignisse $enter1$ und $enter2$, und analog das Vorhaben, den kritischen Abschnitt zu verlassen, durch die Ereignisse $exit1$ und $exit2$ angezeigt. In jedem Kontrollzustand des Systems gilt ein charakteristisches Prädikat für die zu abstrahierende Datenvariable S. Diese Prädikate bilden die Grundlage für eine endliche

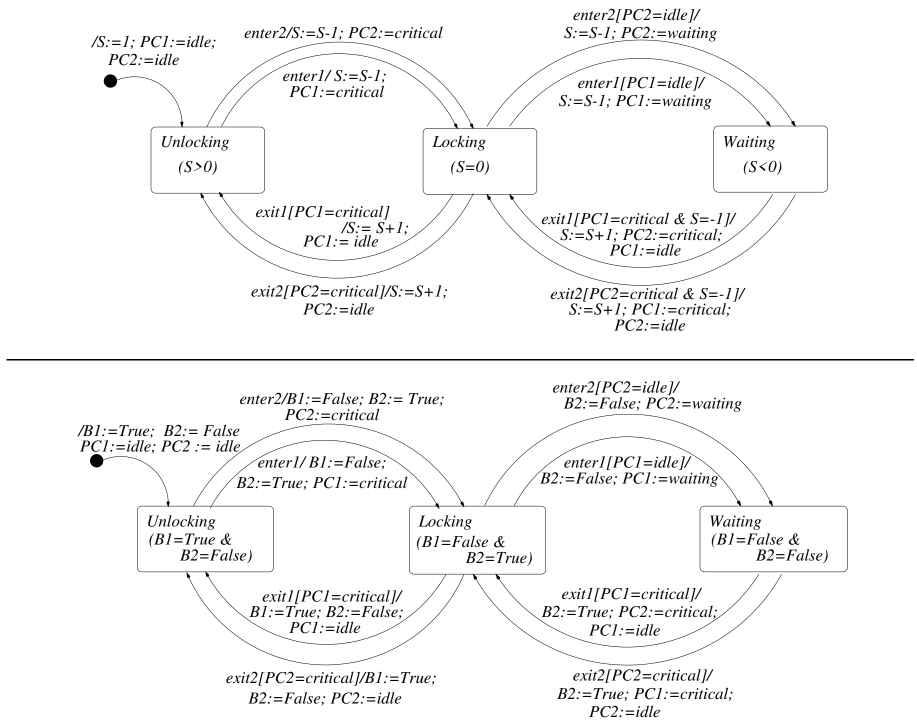

Abbildung 2. Datenabstraktion.

Partitionierung der Domain von S und damit auch für die Konstruktion des abstrakten Systems. Die zentrale Idee bei der Prädikatenabstraktion ist, im abstrakten System boolsche Variablen einzuführen, die angeben, ob die Prädikate, die S charakterisieren, erfüllt sind. In diesem Beispiel ist es ausreichend zwei abstrakte Variablen $B1$ und $B2$ einzuführen, die den Wahrheitswert der Prädikate $S > 0$ und $S = 0$ repräsentieren, da sich $S < 0$ aus der Nichterfülltheit der beiden anderen Prädikate ableiten lässt.

Neben den so ermittelten Zuständen müssen anschließend die Transitionen des abstrakten Modells algorithmisch bestimmt werden. Bei diesem Vorgehen werden die Transitionen als Prädikate über einem Vor- und einem Nachzustand aufgefasst. In [17] wurde gezeigt, dass aus relationalen Verfeinerungsbeziehungen zwischen zwei Transitionssystemen die Existenz einer Galois-Korrespondenz [6] zwischen den beiden Prädikatensystemen folgt. Unter Verwendung der Einbettungsfunktion dieser Galois-Korrespondenz lässt sich ein Prädikat des konkreten Systems in ein Prädikat des abstrakten, und umgekehrt unter Verwendung der Projektionsfunktion ein Prädikat des abstrakten Systems in ein Prädikat des konkreten transformieren. Somit lassen sich die als Prädikate aufgefassten Transitionen des abstrakten Modells mit Hilfe der Einbettungsfunktion aus den Transitionen des konkreten Modells berechnen. Oft ist diese Berechnung allerdings zu aufwendig, so dass auf eine geeignete Approximation zurückgegriffen werden muss.

Schließlich erhalten wir für unser Beispiel das im unteren Teil der Abb. 2 dargestellte Modell. Um beispielsweise eine Sicherheitseigenschaft auf dem konkreten Modell zu zeigen, wird diese analog zu den zweistelligen Prädikaten der Transitionen unter Verwendung der Projektionsfunktion in ein Prädikat des abstrakten Systems transformiert und dann durch Modelchecking verifiziert.

In unserer zukünftigen Arbeit wollen wir im Zuge der Integration eines Theorembeweisers mit einem symbolischen Model Checker die bekannten Abstraktionsansätze um folgende Inhalte erweitern:

1. Wir werden untersuchen, wie das charakteristische Prädikat eines konkreten Zustandes für eine zu abstrahierende Variable maschinengestützt ermittelt werden kann.
2. Hierachische Strukturierungen in den Zustandsautomaten ermöglichen die kompositionale Zerlegung von Verifikationsaufgaben. Um den Model Checker in diesem Sinne effizient auf dem abstrakten Modell einsetzen zu können, wollen wir Abstraktionstechniken entwickeln, die Strukturierungen des konkreten Systems erhalten.

Formale Abstraktionstechniken haben allerdings auch ihre Grenzen. Verifikation in Form von Modelchecking oder Beweisen ist selbst bei diskreten Systemen ab einer bestimmten Größe nicht mehr praktikabel. Insbesondere in objektorientierten Modellen sind Verifikationen, die den Sichtbarkeitsbereich einer Klasse überschreiten, schwierig. Systematisches Testen ist hier eine mögliche Alternative.

3 Testen objektorientierter Softwaresysteme

Der systematische Softwaretest ist ein zentrales Mittel zur Validierung von Programmen. Im klassischen Softwaretest werden dabei Testverfahren unterschieden, die für den Test die Struktur des Programmcodes heranziehen (*White-Box*-Verfahren) und Testverfahren, die auf der Spezifikation eines Programms basieren (*Black-Box*-Verfahren). Ein Grundprinzip des klassischen Softwaretests ist die Verwendung von Testverfahren aus beiden Klassen, da *White-Box*-Verfahren eine fehlerhafte Implementierung der Spezifikation nicht aufdecken können, *Black-Box*-Verfahren anderseits in der Regel nicht zu einer vollständigen Codeüberdeckung führen [16].

Mit der zunehmenden Verbreitung objektorientierter Softwareentwicklung treten die Probleme der Übertragbarkeit klassischer Testverfahren auf objektorientierte Software zu Tage. Die Besonderheiten objektorientierter Software, die die anderen Phasen des Softwareentwicklungsprozesses wie Analyse und Design erleichtern und die Wiederverwendung unterstützen, erschweren den Test. Klassische Testverfahren sind nur bedingt oder nicht anwendbar. Die Probleme lassen sich in erster Linie auf Datenkapselung, Polymorphie und Vererbung zurückführen [20]. Zudem erschwert die zyklische Struktur der Beziehungen zwischen Klassen den Integrationstest, da eine Testreihenfolge für die Klassen nur schwer zu ermitteln ist.

Datenkapselung ist mit dem Verstecken von Informationen (*Information Hiding*) verbunden. Man muss beim Testen zwischen dem innerem versteckten und dem äußerlich sichtbaren Zustand unterscheiden. Oft ist nur anhand des inneren unsichtbaren Zustandes eine Überprüfung des Verhaltens während des Tests möglich, so dass ein Test

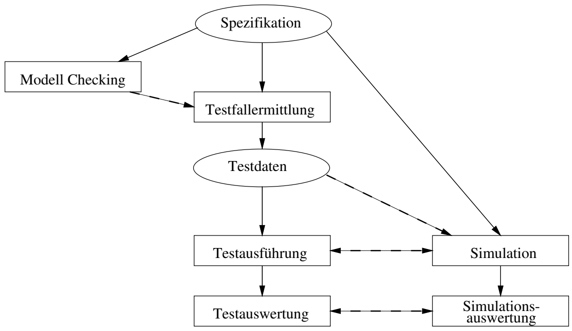

Abbildung 3. Integration von V&V-Techniken

nur ungenügend erfolgen kann. Bricht man die Kapselung für den Test auf, können neue Fehler erzeugt werden, die Implementierungsfehler überlagern.

Polymorphie führt zu verdeckten Kontrollflüssen, weshalb die im klassischen Softwaretest so wichtigen kontrollflussbasierten Verfahren nur bedingt anwendbar sind. Das Prinzip des klassischen Tests, immer Spezifikation und Implementierung zu berücksichtigen, ist somit im objektorientierten Test nur schwer zu realisieren. Ein Test anhand der Spezifikation gewinnt an Bedeutung, es ist also zwingend nötig, dass diese in geeigneter Form vorliegt.

Durch Vererbung muss der Test für alle Unterklassen wiederholt werden, auch wenn die entsprechende Oberklasse vollständig getestet wurde. Durch vererbungsinverse Applikationen werden Methoden der Oberklasse geändert, obwohl sie nicht selbst überschrieben wurden. Durch konforme Vererbung [10] kann zwar der Testaufwand so weit verringert werden, dass die Testdaten des Oberklassentest wiederverwendet werden können und nur für die besondere Eigenschaften der Unterklassen neue Testfälle und -daten erzeugt müssen. Alle in der Oberklasse getesteten Eigenschaften müssen auch in den Unterklassen erneut getestet werden.

Da ein Test anhand der Kontrollflüsse nur schwer möglich ist, wird zur Überdeckungsmessung auf die Tatsache zurückgegriffen, dass Objekte in der Regel einen Zustand besitzen. Die Kriterien der Überdeckungsmessung des Kontrollflusses in Form eines Graphen lassen sich prinzipiell auf die Überdeckungsmessung eines Zustandsgraphen übertragen, so ist die Überdeckung aller Knoten im klassischen Test die Überdeckung aller Anweisungen im Kontrollflussgraphen und im objektorientierten Test die Überdeckung alle Zustände im Zustandsgraphen. Es gibt Ansätze, auch die klassischen Überdeckungsmessungen anhand der Implementierung einzubeziehen, z. B. durch die Verbindung von Kontrollflussgraphen einzelner Methoden mit dem Zustandsgraphen einer Klasse [3].

Automatisierung spielt beim Test insbesondere großer Softwaresysteme eine wichtige Rolle. Dabei ist eine Automatisierung der einzelnen Phasen des Tests (siehe Abb. 3) nur durch eine Spezifikation in einer formalen Form zu erreichen.

Für die Überdeckungsmessung des Tests einer Klasse in der Phase der Testauswertung lassen sich z.B. die Statecharts-Diagramme einer UML-Spezifikation heranziehen.

Auch eine automatisierte Generierung der Testfälle aus der Spezifikation ist möglich. Dazu ist es nötig, neben den Klassen und dem dynamischen Verhalten der Objekte dieser Klassen die Objektinteraktivität zu spezifizieren. Dies ist in UML mit den Interaktionsdiagrammen nur ausschnittsweise möglich. Ein Ansatz zur Automatisierung der Testfallermittlung ist die Integration des Modell Checkings in den Testprozess [8] (vgl. Abb. 3).

Im Bereich der automatisierten Testfallgenerierung wird einer der Schwerpunkte unserer zukünftigen Forschungsarbeit liegen. Weitere Forschungsarbeit dient der Integration von Testen und Simulation (Abb. 3). So können die Software-Bestandteile hybrider Systeme getestet, die Hardware-Bestandteile dagegen simuliert werden. Es sollte möglich sein, die erzeugten Testdaten auch zur Steuerung der Simulation zu nutzen. Darüber hinaus dienen die Ergebnisse aus Testausführung und Simulationsläufen dazu, eine gemeinsame Aussage über die Qualität eines Systems erhalten.

4 Simulation hybrider Systeme

Wie im vorigen Abschnitt erwähnt wurde, sind auch der Analysestrategie *Testen* in Bezug auf die strukturelle Komplexität des Gesamtsystems Grenzen gesetzt. Hier stellt die *Simulation* von Systemen eine weitere Alternative dar. Betrachtet man zudem die zweite Dimension aus Abb. 1, so finden sich nur wenige praktikable Ansätze für Verifikation und Testen kontinuierlicher und hybrider Systeme. Wir möchten uns in diesem Abschnitt auf den Bereich *hybrider Systeme* beschränken.

Interagierende (physikalische) Komponenten mit diskreten und kontinuierlichen Verhalten werden im ingenieurtechnischen Umfeld als hybride Systeme bezeichnet. Typische Vertreter sind digitale Geräte, *zusammen* mit ihren analogen Umgebungen, die sie steuern. Zentral ist in diesem Bereich die Unterscheidung von *realen* Systemen und *Modellen* dieser Systeme, anhand derer Analysen durchgeführt werden können.

Angetrieben durch den Fortschritt in der Entwicklung eingebetteter digitaler Steuerungen, gewinnt die Modellierung und Analyse der hier betrachteten Systeme zunehmend an Bedeutung. Dabei sind klassische rein diskrete bzw. kontinuierlicher Techniken nur für bestimmte Teilaspekte sinnvoll einsetzbar:

Ansätze, die sich ausschließlich auf die Analyse der *diskreten Subsysteme* konzentrieren, haben hier nur eine begrenzte Anwendbarkeit, da im Rahmen der Modellbildung von den Eigenschaften kontinuierlicher Subsysteme abstrahiert werden muss. Andererseits wurden auch für die Modellierung rein *kontinuierlicher Systeme* Beschreibungsmittel, z.B. ACSL [18] oder ASCEND [19], entwickelt, um stückweise stetige Verhalten geeignet repräsentieren zu können. Diese basieren meist auf Algebro-Differentialgleichungssystemen, die durch unterschiedliche Lösungsverfahren behandelt werden können. Die Einbettung der Beschreibung diskreter Phenomäne in diesen Formalismus ist jedoch schwierig.

Erst seit relativ kurzer Zeit stellt die Untersuchung von gemischt kontinuierlich/ diskreten Systemen ein aktives Forschungsgebiet in der Informatik dar. Ziel ist es, eine ganzheitliche Analyse der Systemmodelle zu ermöglichen. Als problematisch erweist sich dabei einerseits die strukturelle Komplexität der Modelle, die von den Beschreibungstechniken geeignete Strukturierungsmittel erfordert. Andererseits sind solche Modelle durch relativ starke Kopplung zwischen ihren Komponenten gekennzeichnet.

Das objektorientierte Paradigma zeigt auch in diesem Gebiet interessante Möglichkeiten auf, diese Probleme zu bewältigen. So führt die Repräsentation physikalischer Komponenten und Eigenschaften in Form von Objekten zu Modellierungsartefakten, die leichter wiederverwendbar sind. Andererseits erlauben objektorientierte Abstraktions- und Strukturierungskonzepte die Dekomposition komplexer Modelle hybrider Systeme in feingranulare Artefakte. In viele Beschreibungstechniken wurden deshalb in den letzten Jahren objektorientierte Prinzipien eingearbeitet (Dymola [7], Smile [15], ZimOO [11]). Die in Abb. 1 dargestellte Einteilung der V&V-Techniken lässt sich auch in diesem Bereich aufrechterhalten, wenngleich einige Techniken eine andere Qualität haben.

In den vergangenen Jahren wurden verschiedene Formalismen zur Spezifikation hybrider Systeme, wie HA [1], EDC [5], ECPN [22] entwickelt, deren Ziel die *formale Verifikation* dieser Systeme ist. Dieser Ansatz beschränkt sich jedoch auf eine kleine Teilmenge von Systemen, da für die meisten mathematischen Probleme in diesem Bereich keine geschlossenen Lösungen existieren, bzw. Methoden bekannt sind, diese zu finden. Somit ist formale Verifikation nur bedingt einsetzbar. Andererseits erfordert *systematisches Testen* hybrider Systeme die Existenz eines Prototypen, also bespielsweise eine Steuerung zusammen mit der realen Anwendungsumgebung. Fehler, die im Rahmen von Testläufen aufgedeckt werden, lassen sich so nur mit erheblichem Kostenaufwand beheben, da ein Validierungszyklus die Rekonstruktion des Prototypen nach sich zieht.

Die *Simulation* technischer Systeme spielt daher mit steigender Komplexität der Anwendungen eine zunehmend wichtigere Rolle. Simulationsexperimente sind hervorragend geeignet, um Auslegungsrechnungen für die Anpassung vorhandener Lösungen an unterschiedliche Rahmenbedingungen durchzuführen. Dennoch ist eine ausschließlich auf Simulation oder Verifikation basierende Analyse solcher Systeme mit Skepsis zu betrachten, da physikalische Ersatzmodelle starke Abstraktionen von der Realität darstellen, die in der Regel an implizite Rahmenbedingungen geknüpft sind. Solche Gültigkeitsbedingungen überlagern sich häufig bei der Vernüpfung verschiedener Submodelle derart, dass die Validität des Gesamtmodells in Frage gestellt ist.

Eine typische Lösung dieser Problematik sind Hardware-in-the-Loop-Techniken (HIL), in denen einzelne (reale) Subsysteme (Controller-HW/SW) in einer simulierten Umgebung getestet werden. Dieser Ansatz stellt eine Mischform zwischen Simulation und Testen dar. Dabei lassen sich die Rollen der Komponenten innerhalb solcher HIL-Tests auch reinterpretieren: Die simulierte Umgebung können auch diskrete und kontinuierliche Komponenten darstellen, die mit realen Subsystemen gekoppelt werden. Im allgemeinen sollte eine gesunde Mischung *aller* drei genannten Strategien verfolgt werden. Welche Strategie den Schwerpunkt der Analyse darstellt, hängt vom Anwendungsbereich ab.

Sowohl Verifikation als auch Simulation setzten die Existenz eines Modells des hybriden Systems voraus, so dass dem *Modellbildungsprozess* insbesondere mit steigender struktureller Komplexität der Modelle eine besondere Bedeutung zukommt. Im Rahmen unserer Arbeiten beschäftigen wir uns neben der Entwicklung von hybriden Beschreibungstechniken unterschiedlicher Abstraktionsniveaus (UML^h [12], ZimOO

[11], Smile [15]) mit der Konzeption einer solchen objektorientierten Methodik [9] für die *vertikale Integration* dieser Techniken, mit dem Ziel, ausführbare Simulationsmodelle zu entwickeln. Vielversprechend erscheint hier die Adaption eines klassischen objektorientierten Software-Entwicklungsprozesses, da sich viele Parallelen zwischen beiden Gegenstandsbereichen ableiten lassen. Im Entwicklungsprozess der Modelle hybrider Systeme kann man drei Hauptaktivitäten identifizieren: Entwurf, Spezifikation und schließlich die Implementierung.

Ausgehend von den *Anforderungen* werden zunächst Modelle in einer geeigneten *Entwurfsnotation* erstellt. Diese Modelle werden dann um Verhaltensbeschreibungen erweitert. Das Ergebnis ist eine *Spezifikation* des Modells. Anschließend kann die Spezifikation durch ein *Simulationsmodell* in einer geeigeneten Simulationsumgebung implementiert werden. Dieses Programm stellt die eigentliche *Simulation* dar und dient letztlich, parametrisiert durch eine *Experimentbeschreibung*, der Gewinnung der *Simulationsergebnisse*. Zwischen diesen Phasen gibt es verschiedene rückkoppelnde Validierungszyklen, die in der Regel iterativ durchlaufen werden.

5 Ausblick

Komplexe Softwaresysteme lassen sich nur durch die Kombination verschiedener Techniken verifizieren und validieren. Modelchecking ist eine wichtige automatische Verifikationstechnik, die aber Testen nicht ersetzen, sondern nur ergänzen kann. Abstraktionstechniken, wie sie zur Zeit für das Modelchecking entwickelt werden, können auch hilfreich sein, um den Detaillierungsgrad von Spezifikationen für das Testen zu reduzieren und so die Testfallerzeugung zu unterstützen. Solche Techniken müssen kompositional sein, um Teile eines Systems unabhängig voneinander verifizieren zu können. Vorbereitende Aufgaben für das Modelchecking und den Test sind Konsistenz- und Vollständigkeitsprüfungen der benutzten Spezifikationen. Hier ist es wichtig, Doppelarbeit zu vermeiden, und die Resultate der einen Techniken als Grundlage für die andere zu verwenden. Simulationstechniken sind im Bereich hybrider Systeme unerlässlich, sie können jedoch auch z.B. für den Integrationstest wertvolle Hilfe leisten: Häufig scheitern Integrationstests an Kosten, weil es sehr aufwendig oder sogar unmöglich ist, Tests auf einem integrierten Hardware- und Softwaresystem durchzuführen (das war z.B. einer der Gründe für den Unfall der Ariane V). Die Simulation von Systemteilen bietet hier die Möglichkeit, eine größere Überdeckung des Gesamtsystems zu erreichen. Dieses Potential sollte sehr viel intensiver ausgenutzt werden.

Literatur

1. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [14], pages 209–229.
2. K. Araki, A. Galloway, and K Taguchi, editors. *First Intern. Conf. on Integrated Formal Methods (IFM'99)*. Springer-Verlag, London, 1999.
3. R. V. Binder. The free-flow graph: Implementation-based testing of objects using state-determined flows. In *8th Annual Software Quality Week*, 1995.

4. M. Broy, F. Huber, and B. Schätz. Auto-Focus – Ein Werkzeugprototyp zur Eentwicklung eingebetteter Systeme. *Informatik – Forschung und Entwicklung*, 14(3):121–134, 1999.
5. Z. Chaochen, A. P. Ravn, and M. R. Hansen. An extended duration calculus for hybrid real-time systems. In Grossman et al. [14], pages 36–59.
6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *4th ACM Symposium on Principles of Programming Languages*, pages 238–252. ACM, 1977.
7. H. Elmqvist. Dymola – user's manual. Technical report, Dynasim AB, Lund, Sweden, 1996.
8. A. Engels, L. Feijs, and S. Mauw. Test generation for intelligent networks using model checking. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1217, pages 384–398. Springer-Verlag, 1997.
9. T. Ernst, C. Klein-Robbenhaar, A. Nordwig, and T. Schrag. Modellierung und Simulation hybrider Systeme mit Smile. *Informatik – Forschung und Entwicklung*, 15(1):33–50, 2000.
10. A. Frick, R. Neumann, and W. Zimmermann. Eine Methode zur Konstruktion robuster Klassenhierarchien. In *Softwaretechnik 96*, pages 16–23, 1996.
11. V. Friesen. *Objektorientierte Spezifikation hybrider Systeme*. PhD thesis, Technical University of Berlin, 1997.
12. V. Friesen, A. Nordwig, and M. Weber. Object-oriented specification of hybrid systems using UML^h and ZimOO. Proceedings of the *ZUM'98*, Berlin, 1998.
13. W. Grieskamp, M. Heisel, and H. Dörr. Specifying safety-critical embedded systems with Statecharts and Z: An agenda for cyclic software components. In E. Astesiano, editor, *Proc. ETAPS-FASE'98*, LNCS 1382, pages 88–106. Springer-Verlag, 1998.
14. R. Grossman, A. Nerode, H. Rischel, and A. Ravn, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.
15. M. Kloas, V. Friesen, and M. Simons. Smile — A simulation environment for energy systems. In A. Sydow, editor, *Proceedings of the 5th International IMACS-Symposium on Systems Analysis and Simulation (SAS'95)*, pages 503–506. Gordon and Breach Publishers, 1995.
16. P. Liggesmeyer. *Modultest und Modulverifikation - State of the Art*. BI-Wissenschaftsverlag, 1990.
17. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1), 1995.
18. Mitchell & Gauthier Associates (MGA) Inc. *Advanced Continuous Simulation Language (ACSL). Reference Manual*, tenth edition, 1991.
19. P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An object-oriented computer environment for modeling and analysis. *Computers and chemical Engineering*, 15:53–72, 1991.
20. P. Rüppel. Besondere Aspekte beim Test objektorientierter Software. In *Info'96: Informationstechnologien für Wirtschaft und Verwaltung, 4. Brandenburger IuK-Tagung*. v. Decker, 1997.
21. Hassen Saïdi and N. Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *Computer-Aided Verification, CAV '99*, LNCS 1633, pages 443–454. Springer-Verlag, 1999.
22. Y. Y. Yang, D. A. Linkens, and S. P. Banks. Modeling of hybrid systems based on extended coloured Petri nets. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 509–528. Springer-Verlag, 1995.