Einbettung mathematischer Techniken in den Systementwurf

Maritta Heisel, Stefan Jähnichen, Martin Simons, Matthias Weber

Technische Universität Berlin* und GMD Forschungsstelle FIRST†

1 Formale Methoden in der Krise

Die Suche nach leistungsfähigen Abstraktions- und Strukturierungskonzepten war eine wesentliche Triebkraft für die Entwicklung höherer Programmiersprachen und Entwurfsmethoden. Es ist unbestritten, daß durch die praktische Anwendung solcher Konzepte eine Verbesserung der Produktivität, eine Verringerung der Fehlerquote sowie eine Unterstützung von Wiederverwendung und Wartung erreicht wurde. Die Aufzählung dieser Vorteile ließe sich weiter fortsetzen; man betrachte in diesem Zusammenhang nur den derzeitigen Erfolg objektorientierter Konzepte. Auch die Entwicklung formaler Methoden war ganz wesentlich durch das Bemühen um konzeptuelle Abstraktion und Strukturierung gekennzeichnet; es ist aber festzustellen, daß die Akzeptanz formaler Methoden in der industriellen Praxis bisher eher kläglich ist. Etwas ironisch wollen wir im folgenden dieses Phänomen als „chronische Krise“ der formalen Methoden bezeichnen.

Heutige Softwaresysteme sind außerordentlich komplex und zudem wird Software zunehmend im sicherheitskritischen Bereich eingesetzt. Dies hat bereits zu verheerenden Fehlfunktionen geführt (wie z.B. die Unfälle mit dem Therac-25 Bestrahlungsgerät [LT93]). Trotzdem wird für die Zukunft mit einer weiter steigenden Komplexität und Verbreitung sicherheitskritischer Systeme gerechnet. Ein unlängst im Scientific American erschienener Artikel über die „chronische Softwarekrise“ dokumentiert anhand einer Reihe von Beispielen, daß die Konstruktion komplexer zuverlässiger Software-Systeme noch weit davon entfernt ist, eine ausgereifte Ingenieursdisziplin zu sein [Gib94]. Die formalen Methoden erheben den Anspruch, zu einer solchen Disziplin beizutragen und auch solche Qualitätsmerkmale zu garantieren, die sich mit herkömmlichen Methoden nicht erreichen lassen. Hierin liegt ganz offensichtlich die Ursache für das andauernde, und in letzter Zeit sogar gestiegene, industrielle und universitäre Interesse an formalen Methoden, trotz ihrer schlechten bisherigen Akzeptanz.

In der Vergangenheit wurden formale Methoden in verschiedenen industriellen und universitären Pilotprojekten ernsthaft eingesetzt (eine übersicht dazu findet sich in [CGR93]). Dies verhalf ihnen jedoch nicht zum Durchbruch. Viele der an solchen Projekten beteiligten Firmen beschränkten die Anwendung formaler Methoden später wieder auf ein

*Forschungsgruppe Softwaretechnik (FR5-6), Franklinstr. 28/29, D-10587 Berlin, Germany. e-mail: {heisel,jaehn,simons,we}@cs.tu-berlin.de

†Rudower Chaussee 5, D-12489 Berlin, Germany.

ihrer gegenwärtigen Anwendbarkeit entsprechendes Maß; etwa durch die Bildung kleiner Spezialistengruppen im Forschungsbereich, die sich auf spezielle kritische Aspekte von Systemen konzentrieren.

Wo liegen die Gründe für den Mißerfolg bei der Einführung formaler Methoden im größeren Stil? Durch eigene Erfahrung [HWW94], sowie durch die Analyse von Erfahrungsberichten [CGR93, HK91, SOF94] sind wir zu der Überzeugung gelangt, daß ein Hauptgrund in der zu breiten Zielsetzung formaler Methoden besteht. Häufig zielen formale Methoden auf eine vollständige und überlegene Technologie zur kompromißlosen Entwicklung korrekter Systeme ab. Zusätzlich wurden gewöhnlich idealisierte Anwendungsbedingungen vorausgesetzt, die jedoch nur in einer universitären Umgebung, in der diese Methoden meistens entwickelt wurden, zu gewährleisten sind. Der monolithische Charakter formaler Methoden erschwert darüberhinaus den gleichzeitigen Einsatz formaler Methoden mit den heute im industriellen Bereich verwendeten Standardtechniken. Trotzdem hat die Forschung über solche Methoden zu vielen nützlichen Resultaten und Techniken geführt; wir halten sie weiterhin für notwendig, aber sie wird unserer Ansicht nach nicht zu rasch einsetzbaren praktischen Methoden führen.

Wir favorisieren eine etwas bescheidener angelegte Integration formaler Techniken in den Entwurfsprozeß. Ausgehend von bewährten klassischen Entwurfsmethoden sollte untersucht werden, an welchen Stellen mathematische Techniken sinnvoll integriert werden können. Die Verwendung solcher Techniken muß einem erfahrenen Ingenieur überzeugend nahegebracht werden können. Sind erst einmal mathematische Techniken punktuell erfolgreich in den Entwurfsprozeß integriert, und ist deren Nutzen durch entsprechende Experimente und Fallstudien belegt, so kann nach weiteren Integrationsmöglichkeiten gesucht werden. Dies kann insbesondere durch die in der ersten Phase gewonnenen Erfahrungen des Entwurfsteams unterstützt werden. Prinzipiell könnte so im Laufe der Zeit eine Methode mit immer mehr formalen Elementen entstehen. Wir möchten in diesem Zusammenhang betonen, daß wir *nicht* vorschlagen, formale Methoden um klassische Anteile anzureichern. Im Gegenteil, unsere Argumentation zielt auf das umgekehrte Vorgehen ab.

Unter Bezug auf unsere einleitenden Bemerkungen möchten wir nun einige Folgerungen aus unserer Position gegenüber formalen Methoden ziehen:

Die chronische Krise der formalen Methoden wird andauern, solange…

…nicht in der Lage sind, mathematische Techniken in den konventionellen Systementwurf einzubetten. Wir glauben, daß eine pragmatische und liberale Einbettung formaler Techniken in konventionelle Methoden ein Potential birgt, das mit der Verwendung von Abstraktionskonzepten in Programmiersprachen und Entwurfsmethoden vergleichbar ist. Die Anwendung formaler Methoden sollte insbesondere an solchen Stellen befürwortet werden, an denen zugrundeliegende Abstraktionen durch die Verwendung von Mathematik sichtbar gemacht und präzise formuliert werden können und damit mathematischer Argumentation zugänglich werden. Solche Anwendungen sollten durch Beispiele und Fallstudien überzeugend belegt werden. In Fällen jedoch, in denen das Beharren auf Formalität lediglich zu einer Komplikation des Entwicklungsprozesses führen würde, sollte auf die Anwendung mathematischer Techniken verzichtet werden. Man betrachte beispielsweise die Verwendung der mathematischen Sprache Z zur Spezifikation von Systemen mit komplexen Datenbeziehungen: nach unserer Erfahrung lassen sich die Ergebnisse vieler Diskussionen über bestimmte Aspekte

der Funktionalität solcher Systeme häufig durch Z-Schemata in eleganter Weise ausdrücken. Positive Erfahrungen dieser Art haben vermutlich zu der vergleichsweise großen Popularität der Z-Notation geführt. Andererseits scheint Z bei weitem nicht so geeignet zur Spezifikation komplexer Kontrollbeziehungen zu sein; hier sollte man andere Notationen wie etwa Statecharts einsetzen.

…sie als eine vollständige Technologie auffassen. Formale Methoden sind sicherlich keine exklusive oder überlegene Vorgehensweise zur Entwicklung sicherheitskritischer Systeme. Aufgrund unserer persönlichen Erfahrungen und der Erfahrungen anderer halten wir es für außerordentlich fragwürdig, ob das gegenwärtige Verständnis von formalen Methoden als einer vollständigen Technologie zu vernünftigen und praktikablen Methoden führen wird. Unserer Ansicht nach verlangt die Komplexität zukünftiger eingebetteter Systeme das ständige Experimentieren mit einer Vielzahl neuer Techniken, einschließlich mathematischer Techniken, zur Beherrschung der Komplexität und zur Verbesserung der Sicherheit. Es ist eine allgemein akzeptierte Tatsache, daß durch die Verwendung formaler Methoden allein nicht *garantiert* werden kann, daß das System sich so verhält wie erwartet bzw. gefordert. Außerdem bieten formale Methoden wenig oder gar keine Hilfe bei der Lösung vieler anderer Fragestellungen, die beim Entwurf sicherheitskritischer Systeme auftauchen. Man betrachte hier nur das Problem, für ein sicherheitskritisches System eine möglichst einfache und durchschaubare Gesamtarchitektur zu finden, die die wirklich sicherheitskritischen Aspekte weitestgehend vereinfacht und lokalisiert. Eine in diesem Sinne transparente Systemarchitektur kann zur Sicherheit eines Systems ebensoviel beitragen wie die Verifikation ausgewählter kritischer Funktionen. Als weiteres Beispiel sei hier noch die Berücksichtigung von Fehlertoleranz beim Systementwurf genannt, durch die ein kontrolliertes Verhalten des Systems auch unter verschiedenen Ausfallsituationen erreicht werden soll.

…wir sie als eine reine Sicherheitstechnologie auffassen. Mathematische Techniken sind für die industrielle Entwicklung von komplexen Systemen jeder Art relevant, nicht nur für sicherheitskritische Systeme. Eine Beschränkung der Erwartungen, die an formale Methoden geknüpft werden, führt zu einem wesentlich grösserem Nutzungspektrum. Als Beispiel sei hier das systematische Testen aufgeführt, das zweifellos für jedes komplexe System unabdingbar ist. Üblicherweise treten dabei die Aktivitäten der Testfalldefinition, der Testdatengenerierung, sowie der Testauswertung auf. Die Spezifikation abstrakter Testfälle in einer geeigneten formalen Notation könnte eine Grundlage dafür bieten, die gewöhnlich sehr aufwendige Testauswertung partiell zu automatisieren, man denke nur an Regressionstests.

…wir versuchen monolithische Entwicklungsumgebungen für sie zu bauen. Werkzeugunterstützung ist für die industrielle Verbreitung mathematischer Techniken unerläßlich. Nach unseren bisherigen Ausführungen dürfte klar geworden sein, daß die Entwicklung einer monolithischen Entwicklungsumgebung wenig sinnvoll ist. Stattdessen sollten lose gekoppelte Werkzeuge gebaut werden und in eine geeignete Gesamtarchitektur eingebettet werden. Notwendige Werkzeuge sind dabei vor allem „mathematische Assistenten'", analog etwa dem System Mathematica, die Wissen über Notationen, Formalismen, und mathematischen Strukturen enthalten, wie sie bei der Systementwicklung benötigt werden.

Im folgenden versuchen wir, unsere Position weiter zu verdeutlichen, indem wir unsere Verwendung mathematischer Techniken in drei gegenwärtig an der Technischen Universität Berlin durchgeführten Projekten beschreiben: die Entwicklung einer Simulationsumgebung für energiewandelnde Systeme, die Formalisierung von Teilen traditioneller Entwurfsprozesse, sowie die Entwicklung einer objektorientierten Methodik zum Entwurf sicherheitskritischer eingebetter Systeme.

2 Formale Methoden für die dynamische Simulation

In einem interdisziplinären Forschungsprojekt wird eine objekt-orientierte und gleichungsbasierte Simulationsumgebung für dynamische Systeme entworfen und implementiert. Gleichzeitig wird sie zur Simulation komplexer energiewandelnder Systeme eingesetzt, um etwa Fragen der Energie-Effizienz und der Umweltbeeinflussung solcher Systeme zu beantworten. Diese Umgebung umfasst eine objekt-orientierte Simulationssprache, einen Übersetzer, ein Laufzeitsystem, dazugehörige numerischer Löser, eine grafische Benutzerschnittstelle sowie eine Komponentenbibliothek, mit deren Hilfe komplex Systeme modelliert werden können. Diese Umgebung wird in enger Zusammenarbeit mit Ingenieuren und Mathematikern entworfen, wobei moderne Methoden und Werkzeuge der Softwaretechnik zum Einsatz kommen, wie zum Beispiel Übersetzerbauwerkzeuge, GUI Konstruktionswerkzeuge und Konfigurations-Management Werkzeuge.

Es ist hierbei nicht unser Ziel, diese Umgebung an sich mittels formaler Methoden zu entwerfen. Wir schlagen vielmehr vor, Elemente formaler Methoden auf die Entwicklung dynamischer Simulationen anzuwenden. Dieser Prozess kann als Spezialfall des Softwareentwicklungsprozesses gesehen werden und er weist in der Praxis auch viele derselben Probleme auf. Auf der anderen Seite ist zumindest die Basis, von der ausgehend man eine Simulation entwickelt, mathematisch gut verstanden.

Ein dynamisches System wird traditionell durch eine Menge von Differentialgleichungen und algebraischen Gleichungen modelliert, die den Zusammenhang zwischen den beobachtbaren Größen des Systems darstellen. Um das modellierte System zu simulieren, wird das Modell in der Regel unmittelbar in eine Simulationssprache umgesetzt. Die notwendigen numerischen Berechnungen werden entweder direkt implementiert oder aber indirekt realisiert, indem numerische Löser des Laufzeitsystems aufgerufen werden. Durch Vergleichen simulierter Werte mit erwarteten, geschätzten oder gemessenen Werten wird das mathematische Modell validiert bzw. verifiziert.

Das mathematische Modell dient somit als Spezifikation für die Implementierung der Simulation. Solange die modellierten Systeme nicht allzu groß sind, ist das beschriebene Vorgehen im Prinzip ausreichend. Werden jedoch große Systeme untersucht, die aus Teilsystemen und tausenden von Gleichungen bestehen, dann ist der Schritt vom mathematischen Modell direkt zur Implementierung zu groß und fehleranfällig. Es wird ein rechnerorientiertes Modell benötigt, daß auch die rechnerorientierten Aspekte eines zu simulierenden Modells in Betracht zieht, die zwar von der Mathematik ignoriert werden, aber von Bedeutung sind, wenn das Modell auf einem Rechner simuliert werden soll.

Unser Ansatz, solche rechnerorientierte Simulationsmodelle zu beschreiben, besteht im Entwurf einer objekt-orientierten Spezifikationssprache für hybride diskret-stetige Systeme. In dieser Sprache wird ein hybrides System gemäß konventioneller

objekt-orientierter Methoden strukturiert, Zustandsvariablen werden benannt und typisiert und die Beziehungen zwischen diesen Größen werden — wie in der mathematischen Beschreibung auch — durch Gleichungen dargestellt. Diese Spezifikationssprache kann als eine Erweiterung von Object-Z [CDD+90] angesehen werden. Eine Spezifikation besteht aus einer Menge von Klassen von denen es drei verschiedene Typen gibt: diskrete, stetige und hybride Klassen. Diskrete Klassen entsprechen den Klassen von Object-Z. Der Zustand einer stetigen Klasse wird durch stetige Variablen charakterisiert, die Funktionen der Zeit sind. Die Zustandsinvariante wird durch Z-Prädikate sowie durch Differentialgleichungen und algebraische Gleichungen ausgedrückt. Ein Model, welches in dieser Sprache ausgedrückt wird, liegt wesentlich näher an einem rein mathematischen Modell als an einem Simulationsprogramm. Lediglich die Struktur des Modells und die Abhängigkeiten zwischen den Teilkomponenten ist nun explizit beschrieben. Nebenbedingungen, die das korrekte Funktionieren einer Teilkomponente einschränken, können formal ausgedrückt werden. Somit kann eine solche formale Beschreibung eines Simulationsmodells einer Komponente unmittelbar als Dokumentation dieser Komponente dienen.

Sobald eine präzise Semantik für diese Spezifikationssprache ausgearbeitet wurde, kann sie uns als Basis für weitere Untersuchungen dienen. Zum einen soll untersucht werden, inwiefern eine automatische Übersetzung einer Modellspezifikation in ein Simulationsprogramm möglich ist und wo numerische Randbedingungen diese Übersetzung beeinflussen. Ausserdem soll untersucht werden, wie der Verfeinerungsbegriff der Modellierungstheorie mit dem Verfeinerungsbegriff der formalen Methoden zusammenhängt. Dies spielt eine Rolle in der adaptiven Simulation. Angenommen man möchte ein Haus als Energieverbraucher modellieren. Es kann entweder als einzelne Einheit modelliert werden, indem man angibt, wieviel Energie es verbraucht in Abhängigkeit von der Zeit. Um aber präzisere Ergebnisse zu haben, kann das Haus in Räume und diese wiederum in Wände und Türen zerlegt werden, wobei für jede dieser Komponenten der eigene Energieverbrauch spezifiziert wird. Schließlich soll untersucht werden, wie die Unabhängigkeit von Teilsystemen deren parallele Simulation für einen gewissen Zeitraum ermöglicht. Die modell-inherente Parallelität ist in der Regel nur noch schwer auf der Ebene eines Simulationsprogramms zu erkennen. Auf der abstrakten, symbolischen Ebene der Spezifikationssparche bietet sich dafür ein besserer Ausgangspunkt.

Die im vorangegangenen Abschnitt genannten Forschungsziele sind bislang lediglich ansatzweise untersucht worden. Was jedoch klar geworden sein sollte ist die Motivation, die hinter diesen Zielen steht: Wir haben versucht Bereiche zu isolieren, in denen formale Techniken vorteilhaft angewendet werden können. Auf der einen Seite versuchen wir, möglichst viele Ergebnisse aus dem Bereich formaler Methoden zu verwenden. Auf der anderen Seite wollen wir in diesem eingeschränkten Anwendungsbereich mit der Weiterentwicklung formaler Methoden experimentieren.

3 Formalisierung traditioneller Entwicklungsprozesse

Ziel dieses Projektes ist es, die Formalisierung einzelner Elemente traditioneller Entwicklungsmethoden möglich zu machen. Dies ist dann sinnvoll, wenn bestimmte *semantische* Eigenschaften eines Produktes positiv garantiert werden sollen. Für Pro-

gramme kann eine solche Eigenschaft die funktionale Korrektheit oder die Einhaltung gewisser Komplexitätsschranken sein. Betrachtet man Spezifikationen, so möchte man z.B. garantieren, daß bestimmte Funktionen invers zueinander sind oder daß ein System gewisse "unsichere" Zustände nicht annehmen kann.

Die Grundidee des hier verfolgten Ansatzes ist, traditionelle Prozesse möglichst unverändert beizubehalten, sie aber in einer Weise zu repräsentieren, die sie mathematischer Argumentation zugänglich macht. Es sollen also, pointiert gesprochen, formale Methoden an traditionelle Prozesse angepaßt werden und nicht umgekehrt. Systementwicklung wird in diesem Zusammenhang als ein *Problemlösungsprozeß* aufgefaßt.

Prozesse als Strategien. Die Formalisierung eines Teilprozesses setzt die Definition dessen voraus, was unter einem Problem genau verstanden werden soll, welches Aussehen eine Lösung haben kann, und wann eine Lösung für ein gegebenes Problem akzeptabel ist. Soll beispielsweise der Übergang von einer Spezifikation zu einem Programm behandelt werden, so sind Probleme im großen und ganzen Ausdrücke einer (formalen) Spezifikationssprache; Lösungen sind Programme in einer Programmiersprache, und eine Lösung ist akzeptabel, wenn das Programm (sei es partiell, sei es total) korrekt bezüglich der Spezifikation ist und/oder andere Qualitätsmerkmale (z.B. minimale Komplexität des gewählten Algorithmus) erfüllt.

Ist der Rahmen, in dem der Problemlösungsprozeß stattfinden soll, auf diese Weise abgesteckt, geht es darum Methoden zur Problemlösung einheitlich zu repräsentieren. Außerdem muß garantiert werden, daß eine mit der Methode entwickelte Lösung auch tatsächlich akzeptabel für das gegebene Problem ist. Zu diesem Zweck wurde ein Wissensrepräsentationsmechanismus, genannt *Strategie*, entwickelt [**?**].

Strategien beschreiben mögliche Schritte, die während einer Entwicklung ausgeführt werden. Wollen wir beispielsweise einen Divide-and-Conquer-Algorithmus gemäß dem Schema

$$\begin{aligned} f(x) \equiv\ & \textbf{if}\ primitive(x)\ \textbf{then}\ directly_solve(x) \\ & \qquad\qquad\qquad\quad \textbf{else}\ \ compose \circ (g \times f) \circ decompose(x) \end{aligned}$$

mit $g = f$ or $g = id$ (die Identität)

für ein gegebenes Programmierproblem entwickeln, so müssen wir geeignete Algorithmen für die im Schema noch unbestimmten Teilfunktionen $directly_solve$, $compose$ und $decompose$ entwickeln, so daß der Gesamtalgorithmus letztendlich korrekt ist. Für diesen Zweck gibt es "Kochrezepte", die mögliche Vorgehensweisen zu diesem Zweck beschreiben.

Für unser Beispiel ist folgendes eine mögliche Vorgehensweise: (i) Konstruiere einen einfachen Dekompositionsoperator $decompose$. (ii) Finde das Kontrollprädikat $primitive$. (iii) Konstruiere den Kompositionsoperator $compose$. (iv) Konstruiere einen Algorithmus für $directly_solve$. Diese Vorgehensvorschrift hilft dann weiter, wenn mit anderen Mitteln entschieden worden ist, daß das gegebene Problem mit einem Divide-and-Conquer-Algorithmus gelöst werden kann. An diesem Punkt setzen Strategien an: Sie repräsentieren die Art von Wissen, wie sie in Textbüchern über Softwaretechnik zu finden ist. Die Kreativität hingegen, die erfahrene Softwareentwickler benötigen, um unter verschiedenen Lösungsideen die vielversprechenste auszuwählen, kann und soll nicht formalisiert werden.

Technisch gesehen ist es der Zweck einer Strategie, eine Lösung für ein gegebe-

nes Entwicklungsproblem zu finden. Dies wird durch *Problemreduktion* erreicht: Das Problem wird gelöst, indem eine Reihe von Unterproblemen generiert werden, aus deren Lösungen die Lösung für das gegebene Problem zusammengesetzt werden kann. Die Unterprobleme werden natürlicherweise auch durch Strategieanwendungen gelöst. Der Prozeß terminiert, wenn die Unterprobleme so einfach geworden sind, daß sie direkt gelöst werden können. Jede Lösung, die im Laufe des Prozesses konstruiert wird, wird natürlich daraufhin überprüft, ob sie akzeptabel ist.

Diese vergröberte Darstellung läßt allerdings die wichtige Tatsache außer acht, daß die generierten Unterprobleme voneinander abhängig sein können. In unserer Divide-and-Conquer-Strategie kann beispielsweise der Kompositionsoperator erst spezifiziert werden, wenn der Dekompositionsoperator bereits bekannt ist. Dieses Phänomen, daß ein Unterproblem erst aufgestellt werden kann, wenn ein anderes bereits gelöst ist, kommt bei der Programmentwicklung häufig vor, so daß naive Ansätze (generiere alle Unterprobleme auf ein Mal, löse sie unabhängig voneinander) hier nicht anwendbar sind.

Damit garantiert ist, daß eine Strategie ein Entwicklungsverfahren vollständig beschreibt, muß ihre Definition folgende Angaben enthalten:

- die Anzahl der generierten Unterprobleme,
- eine Abhängigkeitsrelation auf den Unterproblemen,
- für jedes Unterproblem eine Prozedur, die beschreibt, wie es unter Verwendung des initialen Problems und der Unterprobleme, von denen es abhängt, definiert wird,
- eine Prozedur, die beschreibt, wie die Gesamtlösung zusammengesetzt wird,
- einen Test auf Akzeptabilität der Gesamtlösung,
- optional eine Prozedur, die eine Erklärung produziert, *warum* eine Lösung akzeptabel ist.

Die Erklärungskomponente ist strenggenommen nicht notwendig, erhöht aber das Vertrauen der Anwender in die Ergebnisse, die mit mathematischen Methoden erzielt werden.

Eine generische Systemarchitektur. Ein Wissensrepräsentationsmechanismus wie das beschriebene Strategiekonzept ist natürlich nicht ausreichend für das Ziel, Teile des Softwareentwicklungsprozesses auf formaler Grundlage durchzuführen. Dies wird erst mit geeigneter Maschinenunterstützung möglich. Zu diesem Zweck wurde eine Architektur für Werkzeuge entwickelt, die das Problemlösen mittels Anwendung von Strategien unterstützen [**?**]. Diese Architektur ist *generisch*, d.h. sie ist unabhängig davon, wie Probleme, Lösungen und Akzeptabilität definiert sind. Ihre beiden wichtigsten Komponenten sind die *Strategiebasis*, der *Entwicklungsbaum* und der *Kontrollbaum*, wie in Bild 1 gezeigt.

Die Strategiebasis enthält für jede vorhandene Strategie einen *Strategiemodul*. Dessen Schnittstelle ergibt sich in natürlicher Weise aus den oben angegebenen Bestandteilen einer Strategie. Da Strategiemoduln in uniformer Weise definiert sind, ist es möglich, routinemäßig neue Strategien in ein Unterstützungssystem einzubauen.

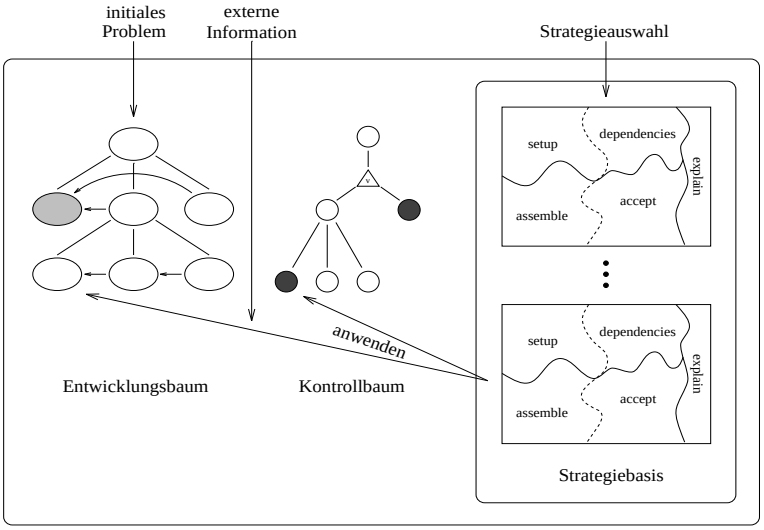

Abbildung 1: Die generische Systemarchitektur

Der Entwicklungsbaum ist eine *Datenstruktur*, die den gegenwärtigen Stand der Entwicklung repräsentiert. Auf diese Weise ist es möglich, zu jedem Zeitpunkt einen Überblick über die Entwicklung zu gewinnen. Der Baum kann durchlaufen werden, um die jeweiligen Probleme und eventuelle Teillösungen zu inspizieren und deren Zusammenhänge zu durchschauen. Dies ist insbesondere dann wichtig, wenn kreative Entwurfsentscheidungen getroffen werden müssen.

Der Kontrollbaum hat nur mit der zukünftigen Entwicklung zu tun. Seine Knoten zeigen auf ungelöste Probleme und bilden die Basis für die Auswahl des nächsten zu bearbeitenden Problems.

Der Entwicklungsprozeß besteht nun aus einer Schleife von Strategieanwendungen, deren jede neue Knoten zum Entwicklungsbaum hinzufügt. Alle generierten (Teil-) Lösungen werden auf Akzeptabilität überprüft. Scheitert dieser Test, wird die zeitlich letzte Strategieanwendung rückgängig gemacht. Ein Zurücksetzen kann auch jederzeit vom Benutzer angestoßen werden.

Die skizzierte Architektur ist in Form eines prototypischen Programmsynthesessystems implementiert [**?**].

Auch mit diesem Projekt folgen wir der eingangs dargestellten Philosophie, formale Methoden nicht in monolithischer Weise, sondern punktuell einzusetzen. Strategien und ihre Anwendung können in traditionelle Prozesse eingebettet werden. Diese werden fast wie bisher ausgeführt, nur daß nun einige Teile formalisiert sind und maschinengestützt ablaufen. Selbstverständlich werden die formalisierten Teile ein gewisses Maß an Expertentum erfordern, doch ist dies ein Preis, unausweichlich gezahlt werden muß, will man semantische Eigenschaften garantieren.

4 Eine Methodik zur Entwicklung sicherheitsrelevanter eingebetteter Systeme

In diesem Projekt sollen bewährte Ingenieurstechniken und neuere mathematische Techniken in einer umfassenden Methodik zur Entwicklung sicherheitsrelevanter eingebetteter Systeme zusammengeführt werden. Ausgangspunkt hierfür ist die marktgängige Technik der Beschreibung des Verhaltens eingebetteter Systeme mit Statecharts, sowie die Verwendung objekt-orientierter Strukturierungskonzepte. Das Anwendungsgebiet dieser Methodik sind integrierte Steuerungssysteme, d.h. Systeme in denen ein Rechner in einer komplexen technischen Umgebung Steuerungsaufgaben wahrnimmt. Durch die zunehmende Komplexität und Verbreitung solcher Systeme wird die Gewährleistung von Sicherheitseigenschaften zu einem immer größeren Problem, zu dessen Beherrschung in Zukunft der stärkere Einsatz mathematischer Techniken erforderlich ist.

Der im Projekt verwendete allgemeine Ansatz zur Modellierung von Sicherheitseigenschaften eingebetteter Systeme liegt in der konzeptuellen Unterscheidung zwischen Sicherheitsanforderungen und (nichtkritischen) funktionalen Anforderungen an das System (Abbildung 2).

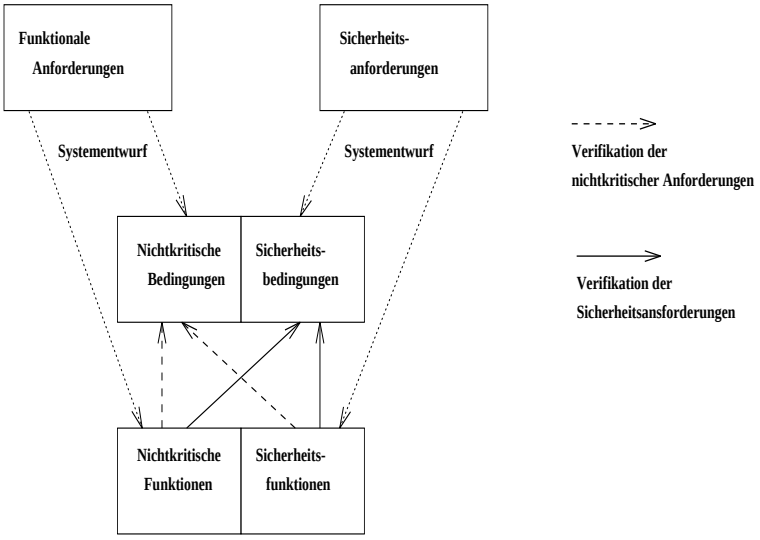

Abbildung 2: Trennung von funktionalen- und Sicherheitsanforderungen

Beispielsweise könnten die Sicherheitsanforderungen an ein Liftsystem die Geschlossenheit der Türen während der Fahrt sowie die Nichtüberschreitung des zulässigen Höchstgewichtes enthalten. Durch den Systementwurf bzw. die Systemarchitektur werden dann die Sicherheitsanforderungen in Bedingungen an die Systemfunktionalität oder in spezielle Sicherheitsfunktionen umgesetzt. Beispielsweise könnten die Sicherheitsbedingungen des Liftsystems fordern, daß der Lift nicht mit der Fahrt beginnen kann wenn das Höchstgewicht überschritten ist oder das die Öffnung der Türen nur bei Halten am einem Stockwerk möglich ist. Die Sicherheitsfunktionen des Liftsystems könnten hingegen die Austattung und elektronische Steuerung von Sicherheitsriegeln sowie eines Gewichtssensor umfassen. Interessanterweise gilt nun aber prinzipiell die gleiche

Aufteilung auch für die Umsetzung der nichtkritischen funktionalen Anforderungen, d.h. auch sie können entweder in tatsächliche Funktionalität oder in Bedingungen an diese Funktionalität umgesetzt werden. Aus diesem Grunde können beide Arten von Anforderungen mit denselben Notationen modelliert werden, bleiben aber während der Systementwicklung, insbesondere während der Verifikation, konzeptuell getrennt.

Die Grundidee der im Projekt vorgeschlagenen Entwurfsmethodik besteht in der Aufteilung der Verhaltensbeschreibung des Systems in mehrere Teile (Abbildung 3): die strukturelle Beschreibung des Systems mit objektorientierten Strukturdiagrammen, die globale Verhaltensbeschreibung der Systemobjekte durch Statecharts, sowie die Beschreibung des lokalen Verhaltens der Systemobjekte mit der mathematischen Sprache Z. Insgesamt erhält man so drei komplementäre, einander ergänzende Sichten auf ein eigebettetes System.

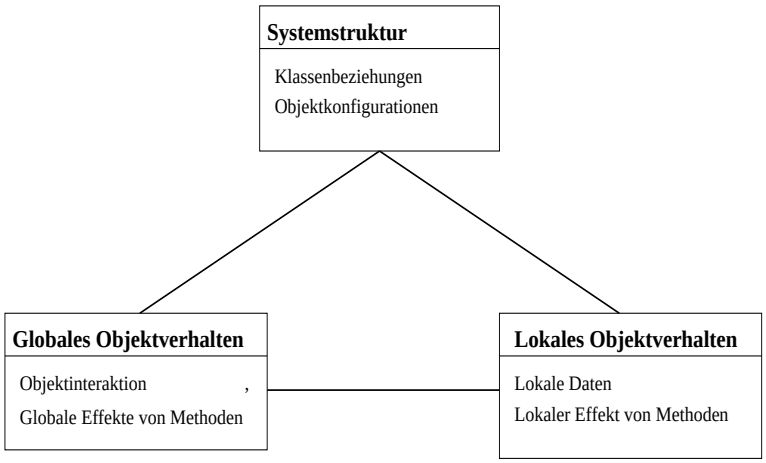

Abbildung 3: Systemsichten und ihre Modellierung

Die strukturelle Sicht beschreibt mit den üblichen Diagrammen der objekt-orientierten Modellierungstechnik die Klassenbeziehungen, d.h. die abstrakte Struktur des Systems, sowie die Systemkonfiguration, d.h. die konkrete Struktur des Systems. Die globale Verhaltenssicht auf ein Objekt beschreibt mit Hilfe von Statecharts dessen Interaktionen mit anderen Systemobjekten, d.h. sie legt fest unter welchen Interaktionen mit den Objekten seiner Umgebung dieses Objekt im Laufe seines Lebenszyklus seine verschiedenen Zustände bzw. Zustandklassen durchläuft. Die lokale Verhaltensicht auf ein Objekt beschreibt mit der mathematischen Notation Z die lokalen Daten des Objekts sowie die lokalen Effekte seiner Operationen.

Auf der Ebene der lokalen Verhaltenssicht auf einzelne Objekte erlaubt die Verwendung der Sprache Z eine streng mathematische Modellierung von Sicherheitsbedingungen, insbesondere solche die bestimmte Zustände von Objekten verbieten. Solche Bedingungen können in Z in Form von Invarianten auf Objektzuständen oder durch entsprechende Vor- und Nachbedingungen der Objektmethoden präzise gefaßt werden. Durch Einsatz von Beweiswerkzeugen für Z können solche Bedingungen darüberhinaus maschinell interaktiv nachgewiesen werden. Auf der Ebene der globalen Verhaltenssicht auf die Objektzustände und Objektinteraktionen können durch die Verwendung von Statecharts Erreichbarkeitsbedungungen sowie andere Lebendigkeitsbedingungen nachgewiesen werden. Als Spezifikationsprache für globale Bedingungen wird

dabei temporale Logik ins Auge gefaßt.

Das Ziel ist die weitgehend automatische Generierung von effizientem Code aus den Modellbeschreibungen. Dies kann im betrachteten Anwendungsgebiet für einen Großteil der Systemfunktionalität erreicht werden, da oft die wesentliche Komplexität in den Statecharts steckt und dagegen das lokale Verhalten nicht sehr komplex ist. Ist das lokale Verhalten dagegen komplex oder nur durch sehr abstrakte Spezifikation transparent modellierbar, so werden an diesen Stellen die Z Spezifikationen schrittweise zu effizienten Datenstrukturen und Programmcode verfeinert.

5 Zusammenfassung

In den vorangegangenen drei Abschnitten haben wir über laufende Projekte unserer Gruppe berichtet. Obwohl diese Projekte untereinander sehr verschiedene Ausrichtungen haben, so ist ihnen doch die im ersten Abschnitt dargestellte Philosophie gemein. Die offensichtlichen wechselseitige Beziehungen zwischen ihnen haben zu einer losen aber fruchtbaren Zusammenarbeit geführt, die sicherlich in der Zukunft noch vertieft werden wird. Wir sind überzeugt davon, daß es nicht sinnvoll und auch nicht möglich ist, eine allumfassende formale Methode zu erfinden, die an die Stelle des traditionellen Entwurfsprozess tritt. Sattdessen versuchen wir, in verschiedene Anwendungsgebieten Teilaspekte dieses Prozesses zu identifizieren, bei denen sich der Einsatz formaler Techniken einfach realisieren läßt und von unmittelbaren Nutzen ist. Für einen solchen Einsatz entwicklen wir dann einen methodische Rahmen sowie geeignete Unterstützungswerkzeuge. Schließlich möchten wir nocheinmal betonen, daß die Forschung auf dem Gebiet der formalen Methoden Hand in Hand mit der Softwaretechnik Forschung einhergehen muß. Insbesondere sollte sich die Forschung darauf konzentrieren, Elemente formaler Methoden mit neuen vielversprechenden Methoden der Softwaretechnik zu kombinieren.

Literatur

[CDD+90] D. Carrington, D. Duke, R. Duke, P. King, G. A. Rose, and G. Smith. Object-Z: An object-oriented extension to Z. In S. Vuong, editor, *Formal Description Techniques, II (FORTE'89)*, pages 281–296. Elsevier, 1990.

[CGR93] D. Craigan, S. Gerhart, and T. Ralston. An international survey of industrial applications of formal methods. Technical Report NISTGCR 93/626, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, 1993.

[Gib94] W. Wayt Gibbs. Software's chronic crisis. *Scientific American*, pages 72–81, September 1994.

[HK91] I. Houston and S. King. CICS Project Report: Experiences and Results from the Use of Z in IBM. In S. Prehn and W. J. Toetenel, editors, *VDM'91 Formal Software Development Methods*, volume 551 of *LNCS*, pages 588–596. Springer-Verlag, 1991.

[HWW94] M. Heisel and D. Weber-Wulff. Korrekte Software: Nur eine Illusion? *Informatik – Forschung und Entwicklung*, 9(4), October 1994.

[LT93] N. G. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *Computer*, 25(7):18–41, 1993.

[SOF94] Special issue on safety-critical systems. *IEEE Software*, January 1994.