Forschungszentrum Karlsruhe
Technik und Umwelt
Wissenschaftliche Berichte
FZKA 5630

Objektorientierte Modellierung am Beispiel eines Mikrosystems zur Messung von Beschleunigungen

K. Lindemann, H. Eggert, W. Süß
Institut für Angewandte Informatik

September 1995

Forschungszentrum Karlsruhe

Technik und Umwelt

Wissenschaftliche Berichte

FZKA 5630

Objektorientierte Modellierung am Beispiel eines Mikrosystems zur Messung von Beschleunigungen

K. Lindemann*), H. Eggert, W. Süß

Institut für Angewandte Informatik

*)von der Fakultät für Maschinenbau der Universität Karlsruhe genehmigte Dissertation

Forschungszentrum Karlsruhe GmbH, Karlsruhe

1995

Als Manuskript gedruckt

Forschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

ISSN 0947-8620

Objektorientierte Modellierung am Beispiel eines Mikrosystems zur Messung von Beschleunigungen

Zusammenfassung

In den letzten Jahren hat sich aus dem Zusammengehen von Mikromechanik und Mikroelektronik die Mikrosystemtechnik entwickelt. Bislang fehlt eine genaue Definition des Begriffs Mikrosystem. Dennoch geht der Trend hin zu miniaturisierten Systemen, die so komplex sind, daß sie ohne eine geeignete Rechnerunterstützung nicht wirtschaftlich entworfen werden können. Insbesondere muß dabei die Zusammenarbeit von Fachleuten aus ganz unterschiedlichen Disziplinen unterstützt werden.

Die Unterstützung des Entwurfs wird erreicht durch die rechnergestützte Modellierung des zu entwickelnden Systems. Bereits in der Anfangsphase des Entwurfs werden wesentliche statische und dynamische Eigenschaften des Gesamtsystems durch die Beschreibung von vereinfachten Modellen der Komponenten des Systems abgebildet. Verschiedene methodische Ansätze werden diskutiert und auf ihre Anwendbarkeit untersucht.

Die gewählte Methodik der Modellierung besteht in einer Beschreibung mit einer objektorientierten Wissensrepräsentation. Die Vorteile dieses Ansatzes bei der Modellierung von Mikrosystemen werden dabei herausgearbeitet. Neben anderen sind hier die Modellierung von statischen und dynamischen Eigenschaften innerhalb einer Methode, die Unterstützung bei der Zerlegung des Systems in seine Komponenten und die Verfügbarkeit von kommerziellen Werkzeugen zu nennen. Für die Durchführung der Modellierung wird, aufbauend auf einer Expertensystemshell, ein Werkzeug entwickelt, welches die Erstellung eines objektorientierten Modells unterstützt.

Anhand einer prototypischen Implementierung eines Mikrosystems zur Messung von Beschleunigungen, welches am Forschungszentrum Karlsruhe realisiert wurde, wird die prinzipielle Realisierbarkeit gezeigt. Dabei können die wesentlichen Eigenschaften des realen Systems in der Modellierung abgebildet werden. Die Datenverarbeitung als wesentliche Systemkomponente kann durch die Integration der Software des realen Systems, welche ebenfalls im Rahmen dieser Arbeit entwickelt wurde, modelliert werden.

Einige mögliche Modifikationen an Komponenten- und Systemdesign des Systems werden beschrieben und die Auswirkungen auf die bestehende Modellierung diskutiert, um zu zeigen, wie das Modell als Grundlage bei einer Weiterentwicklung des Systems dienen kann.

Ein Ausblick zeigt, daß sich die in dieser Arbeit beschriebene Vorgehensweise mit einem bereits vorher beschriebenen Entwurfsablauf für Mikrosysteme vereinbaren läßt. Abschließend werden einige Erweiterungen des Modellierungstools und seine Integration mit anderen Arbeiten aufgezeigt.

Object-oriented modelling demonstrated for a microsystem for acceleration measurement

Abstract

During the last years, microsystem technology emerged from the combination of both the fields of micromechanis and microelectronis. Although a precise definition of the term micro system has not yet been agreed upon, there is a trend towards miniaturized systems, which, due to their complexity, cannot be designed economically without computer based tools. Especially the cooperation of specialists of different disciplines must be supported.

Support of the design process is achieved by implementing a computer based model of the system under development. Starting in the early stages of the design process, the static and dynamic properties of the entire system are described by means of simplified models of the system components. Several methods are examined and their applicability is discussed.

One way of modelling the system is to use a object oriented knowledge representation. The advantages of this approach are shown. Especially the modelling of static and dynamic properties within one method, the support of decomposing the system in its components and the availability of commercial tools are worth mentioning. To carry out the modelling, a tool is developed which is based on an expert system shell.

A microsystem for acceleration measurement, which has been developed at the Forschungszentrum Karlsruhe, is being modelled to show the feasibility of this approach. The essential properties of the real system can be found in the model. The data processing as a vital component is brought into the model by including the software of the real system, which also has been developed as part of this thesis.

Possible modifications of components and the system design are described and their influence on the model is discussed to demonstrate how the model can serve as a base for further development of the system.

In a short outlook it is shown that the approach described in this thesis can be combined with a design scheme for micro systems which has developed earlier. Finally a few possible improvements of the modelling tool and its relation to other works is shown.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Unübersehbar hat die Mikroelektronik mit ihrer Durchdringung weiter Bereiche der Technik das tägliche Leben der Menschen verändert. Fragt man nach den Ursachen für ihren Erfolg, so sind folgende Punkte zu nennen:

- Die Realisierung von Funktionalität auf engstem Raum bei geringem Energieverbrauch durch miniaturisierte Bauelemente.
- Niedrige Kosten der Endprodukte trotz hoher Entwicklungsaufwendungen durch
  - parallele Fertigung (batch-processing) mittels Photolithographie und dadurch mögliche Massenproduktion.
  - rechnerunterstützten Entwurf und Fertigung.
- Neue, vorher nicht oder nur schwer realisierbare Produkte wie Taschenrechner, Computer, Unterhaltungselektronik etc.

In den letzten Jahren hat man erfolgreich damit begonnen, diese Vorteile der Mikroelektronik auf andere Gebiete zu übertragen, wobei primär die Mechanik, Optik, Sensorik und Aktorik zu nennen sind. Vorwiegend wird als Basismaterial Silizium eingesetzt (Si-Technik), weil hier das gesammelte Wissen, Bearbeitungstechniken und Fertigungsanlagen der Mikroelektronik verwendet werden können. Außerdem besitzt Silizium eine Reihe von interessanten mechanischen Eigenschaften, wie ein mit Stahl vergleichbares Elastizitätsmodul bei wesentlich geringerer Dichte [Petersen82]. Als ein Beispiel für Mikromechanik und Mikrosensorik seien hier in Silizium-Technik hergestellte Beschleunigungssensoren genannt [Roylance79]. Die Si-Technik verwendet eine Reihe von Strukturierungstechniken, u.a. das naßchemische anisotrope Ätzen und die Oberflächenmikromechanik. Neben der Si-Technik hat sich das im Kernforschungszentrum Karlsruhe entwickelte LIGA-Verfahren [Becker85][Menz91] in der Mikromechanik und -optik etabliert, weil es der Mikrostrukturierung neue Materialien, freie Formgebung in der Ebene und große Strukturhöhen bei kleinen lateralen Abmessungen erschließt. Beispiele für in LIGA-Technik hergestellte Bauteile sind Beschleunigungssensor [Burbaum91] oder Mikromotor [Wallrabe92].

Aus der Kombination der verschiedenen Mikrotechniken hat sich seit Ende des letzten Jahrzehnts die Mikrosystemtechnik entwickelt, welche Komponenten der Mikroelektronik, Mikromechanik und/oder Mikrooptik monolithisch oder mit Hilfe von Aufbau- und Verbindungstechniken integriert. Abbildung 1 zeigt den schematischen Aufbau eines vollständigen Mikrosystems mit den wesentlichen Bestandteilen Sensorfeld, Aktorfeld, Datenverarbeitung und Schnittstelle nach außen. Dabei ist zu betonen, daß eine bloße Kombination von mikrotechnisch hergestellten Komponenten nicht hinreichend ist, um ein Mikrosystem zu bilden. Wesentlich ist die intelligente Verknüpfung von einzelnen Funktionen zu einer Gesamtsystemfunktion. So kann durch ein Einsatz von Sensorarrays gleicher Sensoren mit einer intelligenten Datenauswertung durch Redundanz eine Zuverlässigkeit und Genauigkeit erzielt werden, die einzelnen Sensoren überlegen ist. Die bei der Herstellung eingesetzten Mikrotechniken sind die Voraussetzung dafür, daß trotz einer hohen Zahl von Einzelkomponenten Mikrosysteme potentiell billiger sind als herkömmliche Sensoren.

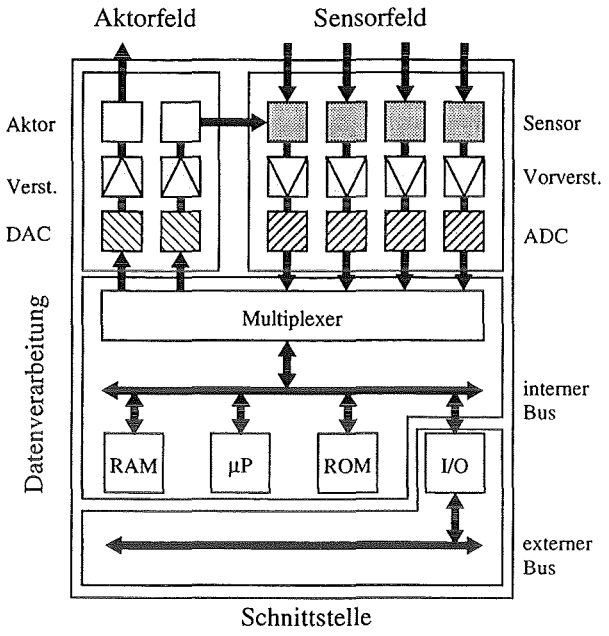


Abb. 1 Schematische Darstellung eines vollständigen Mikrosystems nach [Menz93a], leicht modifiziert. (µP: Mikroprozessor, I/O: Ein-/Ausgabe)

Mikrosysteme haben noch keine Marktreife erlangt, sind allerdings Gegenstand intensiver Forschungstätigkeit. Beispiele zu Vorstufen von Mikrosystemen und in der Entwicklung befindlichen Mikrosystemen sind:

- Ein Chip, bei dem 7 verschiedene Sensorfunktionen, u.a. Druck-, Beschleunigungs- und Infrarot-Messung, auf einer Fläche von $8 \times 9\ mm^2$ untergebracht sind [Polla86]. Allerdings werden die Sensorfunktionen nicht dazu verwendet, eine Systemfunktion zu realisieren. Eine Datenverarbeitung fehlt ebenfalls.
- Ein Herzkatheter [Manoli91], der mehrere Blutparameterwerte mißt. Mehrere Chips, die jeweils eine Sensorfunktion zusammen mit einem Verstärker realisieren, werden auf ein Polyimid-Tape gebondet und bilden so ein Multi-Chip-Modul. Die Meßwerte werden im System digitalisiert, eine Verarbeitung erfolgt jedoch erst extern.
- Ein Mikrosystem zur Messung von Beschleunigungen [Strohrmann93a], welches Beschleunigungen in 2 Raumrichtungen mißt, Offsets und Nichtlinearitäten der Sensoren korrigiert, eine Datenverarbeitung im System durchführt und die verarbeiteten Daten über eine serielle Schnittstelle an einen externen Rechner weitergeben kann.

Die Entwicklung der Mikrosystemtechnik ist noch in ihrer Anfangsphase; die technologischen Grundlagen werden ständig erweitert, wodurch neue Mikrosystemkomponenten realisierbar sind. Daher sind die möglichen Anwendungsgebiete noch nicht voll zu überblicken. Gebiete mit einem wachsenden Bedarf an billiger Sensorik wie die Umweltüberwachung oder die Automobiltechnik werden von der Mikrosystemtechnik ebenso profitieren, wie Anwendungen in der Medizintechnik, bei denen komplexe Meß- und Regelungsaufgaben auf engstem Raum gelöst werden müssen (z.B. künstliche Bauchspeicheldrüse).

Die Definition eines systematischen Vorgehens beim Entwurf und der Entwicklung von Mikrosystemen ist bis jetzt noch nicht abschließend erfolgt. Ein erster Ansatz für einen Entwurfsablauf für Mikrosysteme findet sich in [BMFT93]. Die einzelnen Phasen des Entwurfsablaufs erfordern eine Unterstützung durch rechnergestützte Werkzeuge. Dabei ist zu unterscheiden zwischen dem Entwurf von Mikrosystemkomponenten auf der einen und gesamten Mikrosystemen auf der anderen Seite. Während bei den Komponenten im Bereich der Mikroelektronik aufgrund langjähriger Entwicklung entsprechende Werkzeuge zur Verfügung stehen, ist dies in den anderen Teilgebieten nur eingeschränkt der Fall. Die Entwicklung solcher Werkzeuge erfordert einen erheblichen Aufwand. Ein Beispiel aus dem Bereich der Mikromechanik ist MEMCAD [Senturia92], welches aus einer Fertigungsprozeßbeschreibung ein dreidimensionales Modell der mikromechanischen Struktur erstellt, dessen mechanische Eigenschaften dann mit FEM berechnet werden; ebenso können elektrische Eigenschaften (Ladungsverteilung auf Oberfläche und Kapazität) bestimmt werden. Das System ist sehr umfangreich, es werden, wo möglich, für Teilaufgaben kommerzielle Programmpakete (mechanisches CAD, FEM-Paket, relationales Datenbanksystem) eingesetzt. Trotzdem deckt es nur einen Teilbereich der Entwicklung von mikromechanischen Strukturen ab, z.B. wird die Reibung von sich gegeneinander bewegenden Strukturen nicht berechnet.

Eine umfassende Entwurfsumgebung zur Entwicklung von Mikrosystemen wurde bisher nicht verwirklicht. Sie müßte sowohl die Entwicklung der Komponenten als auch des Systems als Ganzes unterstützen, und dies über alle Entwicklungsphasen (Spezifikation, Design, Simulation, Test) hinweg. Da die Realisierung eines derart komplexen Werkzeugs nicht machbar erscheint, wird in [BMFT92] eine Strategie vorgeschlagen, bei der zwischen Werkzeugen für den Komponentenentwurf und dem Systementwurf unterschieden wird, wobei beim Systementwurf von komponentenspezifischen Problemen abstrahiert wird. Ein Vorschlag für eine Spezifikations- und Systementwurfsumgebung für Mikrosysteme findet sich in [Bortolazzi93a]. Um den Entwurf von Komponenten mit einzubeziehen, ist ein Kopplung zwischen einer Systementwurfsumgebung und Komponentenentwurfswerkzeugen erforderlich. Dieses Vorgehen wird auch als "Meet in the Middle"-Strategie bezeichnet. Aktuelle Forschungsvorhaben, die sich mit der Realisierung von Entwurfswerkzeugen und Fragen der Entwurfsproblematik im Bereich der Mikrosystemtechnik beschäftigen, sind u.a. die beiden BMFT-Verbundvorhaben "Untersuchung zum Entwurf von Mikrosystemen" und "Methoden- und Werkzeugentwicklung für den Mikrosystementwurf".

Die Realisierung von Werkzeugen für den System- und Komponentenentwurf ist nur langfristig möglich. Allerdings ist auch danach damit zu rechnen, daß Teilbereiche der Entwicklung von Mikrosystemen ohne ihre Unterstützung durchgeführt werden müssen, weil die Anpassung der Entwurfsumgebungen an die ständige Weiterentwicklung der technologischen Grundlagen der Mikrosystemtechnik nur mit zeitlicher Verzögerung geschehen kann.

Man muß daher davon ausgehen, daß der Entwurf von Mikrosystemen jetzt und in Zukunft nicht nur von einzelnen oder wenigen Personen durchgeführt wird, die allein mit Hilfe eines Entwurfswerkzeugs die komplexen Probleme des Entwurfs von Mikrosystemen lösen können. Vielmehr wird dies die aktive Zusammenarbeit von Experten aus sehr verschiedenen Fachrichtungen (Fertigungstechnik, Maschinenbau, Elektronik, Informatik, Chemie, Optik, Systemtechnik etc.) erfordern. Rechnerunterstützung des Entwurfs hat dabei auch die Aufgabe, die Kommunikation zwischen den Fachgebieten zu ermöglichen, damit die gegenseitigen Abhängigkeiten von Komponentenentwurf und Systementwurf für alle an der Entwicklung des Systems Beteiligten deutlich werden.

Dies ist deswegen nicht von vornherein gegeben, weil die an der Entwicklung Beteiligten, geprägt durch ihr Wissen auf dem jeweiligen Fachgebiet, tendenziell ihre eigene Sicht des Systems haben und daher unterschiedliche Lösungsansätze zur Realisierung der Systemfunktion(en) verfolgen werden. Alle an der Entwicklung eines Mikrosystems Beteiligten sind aber gezwungen, in Systemen zu denken [Rauch89]. Die Unterstützung dieses Systemdenkens ist daher eine wichtige Aufgabe, die eine rechnergestützte Entwurfshilfe erfüllen muß.

1.2 Zielsetzung

Diese Entwurfshilfe soll darin bestehen, daß bereits in einer frühen Phase des Entwurfs mit einer *Modellierung* des geplanten Systems begonnen wird. Unter Modellierung wird dabei die Erstellung eines rechnerbasierten *Modells* verstanden, welches alle zum Verständnis des geplanten Systems notwendigen Informationen enthält. Das Modell kann in einen statischen und einen dynamischen Teil aufgeteilt werden.

Der statische Teil umfaßt alle das System und seine Komponenten charakterisierenden Parameter, welche Auswirkungen auf das Design des Gesamtsystems haben. Solche Merkmale sind z.B. Informationen über die Geometrie von Komponenten oder Eigenschaften der verwendeten Materialien. Dabei sind nicht nur die für ein geplantes System spezifizierten Parameterwerte von Bedeutung. Ebenso wichtig ist es, auch weitergehende Informationen zu erfassen, z.B. in welchen Grenzen ein bestimmter Parameter variiert werden darf und welche Randbedingungen dieses Intervall festlegen. Dieser Teil des Modells hat also auch den Charakter einer Spezifikation vor der Umsetzung in ein reales System und Dokumentation nach dessen Fertigstellung.

Im dynamischen Teil wird das Verhalten des Systems unter Einwirkung externer Einflüsse und seine Interaktion mit der Umwelt modelliert. Dieser Teil des Modells hat den Charakter einer ausführbaren Spezifikation bzw. Simulation, wobei das Verhalten des realen System angenähert wiedergegeben wird. Wichtig ist hierbei, daß das System nicht nur im Sinne einer "black box" modelliert wird, bei der das Zustandekommen des Verhaltens unklar bleibt. Vielmehr muß sich das Verhalten des gesamten Systems aus dem Zusammenwirken seiner Komponenten ergeben.

In einer frühen Entwurfsphase kann die Modellierung auf einer abstrakten Ebene beginnen. Wenn der Systementwurf konkretisiert oder aufgrund von Anforderungsänderungen modifiziert wird, wird das Modell ebenfalls geändert. Am Ende eines Entwurfs soll man im Idealfall in der Lage sein, einem Unbeteiligten das Funktionieren des Systems und wesentliche Designentscheidungen anhand des Modells zu erläutern. Nach der Realisierung des Systems kann es als Ausgangspunkt für die Entwicklung von Varianten verwendet werden.

Ziel der vorliegenden Arbeit ist es, an einem konkreten Beispiel zu zeigen, wie eine rechnergestützte Modellierung von Mikrosystemen auf der Systemebene durchgeführt werden kann, da es hierzu bis jetzt noch keine etablierten Ansätze gibt. Dazu wird eine geeignete Methode entwickelt bzw. ausgewählt und ein Werkzeug auf der Basis dieser Methode zur Durchführung der Modellierung konzipiert und implementiert. Anschließend wird mit diesem Werkzeug die Modellierung eines Mikrosystems durchgeführt und das Modell mit dem realen System verglichen. Dies geschieht in mehreren Schritten.

1.3 Durchführung

Da die Mikrosystemtechnik noch eine sehr junge Disziplin ist, ist es nicht möglich, anhand von Erfahrungen mit bereits realisierten Prototypen abzuleiten, welche konkrete Anforderungen an das Modell zu stellen sind. Es werden daher in Kapitel 2 Kriterien angegeben, an denen Mikrosysteme gemessen werden können, um eine Abschätzung zu ermöglichen, mit welchen Problemen man typischerweise bei der Modellierung konfrontiert werden wird. Auf der Grundlage dieser Bewertung von Mikrosystemen werden danach Forderungen aufgestellt, welche die zur Modellierung eingesetzten Methoden und das Werkzeug erfüllen müssen. Wesentliche Gesichtspunkte dabei sind die Einbeziehung des statischen und des dynamischen Teils des Modells und die Integration des Modellierungswerkzeugs mit anderen im Rahmen des Mikrosystementwurfs verwendeten Werkzeugen. Verschiedene Methoden werden daraufhin untersucht, inwieweit sie den aufgestellten Anforderungen genügen.

In Kapitel 3 werden die Methoden erläutert, welche für eine Modellierung geeignet sind. Dabei wird gezeigt, daß ein objektorientierter Ansatz die aufgestellten Forderungen erfüllt. Danach wird ein Werkzeug prototypisch implementiert, mit dem die Modellierung des Mikrosystems praktisch durchgeführt wird.

Danach wird in Kapitel 4 beispielhaft durch die Modellierung eines Mikrosystems gezeigt, wie eine Umsetzung der Methode und eine Anwendung des Werkzeugs erfolgen kann. Dabei handelt es sich bei dem betrachteten System um das oben genannte Mikrosystem zur Messung von Beschleunigungen, welches parallel zu dem Modell im Kernforschungszentrum Karlsruhe entwickelt wurde. Die parallele Entwicklung eines realen Systems erlaubt es, Aspekte der Modellierung aus mehreren Phasen der Mikrosystementwicklung zu behandeln. Dazu ist es notwendig, die Datenverarbeitung des realen Systems im Rahmen dieser Arbeit zu implementieren.

Um die Flexibilität bei der Erweiterung eines Modells zu zeigen, werden danach in Kapitel 5 einige mögliche Erweiterungen des Mikrosystems betrachtet und die notwendigen Änderungen am Modell diskutiert.

Eine Zusammenfassung der durchgeführten Tätigkeiten und eine Diskussion der erreichten Ziele sowie der möglichen Weiterentwicklungen beschließen die Arbeit.

2 Voraussetzungen für die Modellierung von Mikrosystemen

Im folgenden soll untersucht werden, mit welchen Methoden eine Modellierung von Mikrosystemen auf Systemebene durchgeführt werden kann. Dazu wird zunächst der Begriff Mikrosystem untersucht und Kriterien angegeben, an denen potentielle Mikrosysteme gemessen werden können. Daraus werden Anforderungen an die Modellierungsmethoden abgeleitet. Anforderungen an ein Modellierungswerkzeug ergeben sich durch eine Diskussion, welche Hilfsmittel einem Anwender zur Verfügung gestellt werden müssen. Anschließend werden einige bekannte Methoden auf ihre Eignung zur Modellierung hin überprüft.

2.1 Kriterien für die Bewertung von Mikrosystemen

Um die Durchführung einer Modellierung von Mikrosystemen zu untersuchen, muß geklärt sein, was Mikrosysteme sind und wodurch sie sich definieren. Zur Zeit fehlt noch eine allgemein verbindliche Definition des Begriffs. Einerseits finden sich in der Literatur Begriffe wie intelligenter Mikrosensor oder Mikroroboter, die teilweise anstelle von Mikrosystem verwendet werden, andererseits bezeichnet das Schlagwort "Systems on Chips" [Wieder92] keine Mikrosysteme, sondern den Trend in der Mikroelektronik, Speicher- und Logikfunktionen in einem Baustein zu vereinen. Mikrosysteme lassen sich an folgenden Kriterien messen:

- Basistechnologie
- Art und Zahl der Komponenten
- Systemcharakter
- fertigungstechnischer Integrationsgrad
- Größe

Diese Kriterien werden jetzt kurz diskutiert.

Basistechnologie
Bei der Herstellung von Mikrosystemen und ihren Komponenten werden eine Reihe von Technologien eingesetzt, wie die Si-Technologie oder die LIGA-Technik, die ihrerseits wieder aus einer Vielzahl von Verfahren bestehen, wie Lithographieverfahren, Ätztechniken, Dünnschichttechnik, etc. Wichtiger als die verwendete Technologie ist die dahinterstehende Philosophie, daß bei der Fertigung Bearbeitungsschritte auf einer großen Zahl von miniaturisierten Systemen oder zumindest Komponenten gleichzeitig ausgeführt werden, was erst eine wirtschaftliche Fertigung in größeren Stückzahlen ermöglicht.

Auf die Modellierung des Systems hat die Basistechnologie nur indirekt Auswirkungen durch die Randbedingungen (wie kleinste realisierbare Strukturbreiten etc.), die sie für die Herstellung der Komponenten setzt.

Komponenten von Mikrosystemen
Abbildung 1 zeigte bereits eine schematische Darstellung eines vollständigen Mikrosystems. Man erkennt eine Vielzahl von in ihren Eigenschaften sehr unterschiedlichen Komponenten: Sensoren, Aktoren und verschiedene elektronische Bauteile (analog und digital). Der Mikroprozessor führt die Software aus, welche die Datenverarbeitung realisiert und über die Schnittstelle mit anderen Systemen kommuniziert.

Die Aufteilung der Funktionalitäten auf die im System vorhandenen Komponenten kann allerdings zwischen Mikrosystemen stark variieren, oftmals werden einige Komponenten ganz fehlen. Die Funktionen von ROM, µP, I/O können in einem Mikrocontroller zusammengefaßt werden, der Multiplexer kann direkt hinter den Verstärkern plaziert sein. Aktoren, besonders diejenigen, welche nicht nur auf systeminterne Komponenten, sondern auf einen externen physikalischen oder chemischen Prozeß einwirken, befinden sich größtenteils noch im Forschungsstadium, und sind folglich in vielen Mikrosystemen noch nicht vorhanden. Für viele Anwendungen sind Aktoren nicht erforderlich; man kann hier von Mikro-Meßsystemen sprechen. Je nach Zahl und Art der Sensoren sowie der vom System zu erfüllenden Aufgabe kann ein Mikroprozessor für die Datenverarbeitung überdimensioniert sein und wird z.B. durch einen programmierbaren Logikbaustein ersetzt. Bei der Integration eines Prozessors in das System muß auch die Software als Komponente entwickelt werden. Dies hat direkte Auswirkungen auf den Entwurf auf Systemebene, da die Systemfunktion nach außen im wesentlichen durch die Software bestimmt wird.

Eine hohe Zahl von Einzelkomponenten bedingt auch eine hohe Komplexität des gesamten Systems und motiviert damit die Notwendigkeit einer Modellierung auf Systemebene. Die Heterogenität der Komponenten stellt eine große Herausforderung für die Modellierung dar, welche ihre sehr unterschiedlichen Eigenschaften und Funktionen geeignet darstellen muß. Dieser Aspekt unterscheidet die Mikrosystemtechnik von Teilgebieten wie der Mikromechanik oder Mikroelektronik. Erschwerend kommt hinzu, daß durch die Weiterentwicklung der Basistechnologien ständig neue Arten von Komponenten herstellbar sind, welche auch modelliert werden müssen.

Systemcharakter von Mikrosystemen
Nach [Rauch89] wird bei einem Mikrosystem primär die Gesamtfunktion optimiert, weniger die Teilfunktionen für sich. Dahinter steht der Gedanke, daß eine Optimierung der Teilkomponenten für sich in der Summe aufwendiger ist als die Optimierung der Systemfunktion. Komponenten- und Systementwurf können daher nicht unabhängig voneinander erfolgen. Allerdings sind viele kommerziell verfügbare mikroelektronische Komponenten bereits auf eine bestimmte Funktion hin optimiert, so daß ihr Einsatz in einem Mikrosystem keinen Mehraufwand bei der Entwicklung verursacht. Eine starke Kopplung zwischen Komponentenentwurf und Gesamtsystementwurf bedeutet auch einen Verlust an Modularität, was sich negativ bei einer Weiterentwicklung auf der Basis eines bestehenden Systems auswirkt.

Noch enger faßt [Kroy89] den Begriff Mikrosystem. Danach kann erst dann von Mikrosystemen gesprochen werden, wenn die Gesamtfunktion nur über das Zusammenwirken aller Teile optimiert wird, d.h. durch erhöhten Aufwand bei der Verbesserung von einzelnen Komponenten kann die Gesamtfunktion prinzipiell nicht optimiert werden.

Ein anderer Aspekt ist die Systemfähigkeit. Mikrosysteme werden in vielen Bereichen (z.B. Automatisierungstechnik) im Verbund mit anderen Systemen zur Anwendung kommen. Dafür müssen sie nach außen eine klar definierte oder sogar genormte Schnittstelle zur Verfügung stellen. Die in [Brignell89] für intelligente Sensoren aufgestellte Forderung, daß die Komplexität des Systems nach außen hin nicht sichtbar werden darf, gilt ebenso für Mikrosysteme. Dies bedeutet, daß Funktionen wie Fehlerkorrektur der Sensoren oder Überwachung des Systems im System selber realisiert werden müssen.

Auch der Systemcharakter trägt dazu bei, daß sich die Komplexität von Mikrosystemen erhöht und macht eine Modellierung in der Entwurfsphase erforderlich.

Fertigungstechnischer Integrationsgrad
Auf den ersten Blick scheint eine monolithische Integration aller Komponenten in einem Chip wünschenswert zu sein. Sie würde erlauben, den Vorteil einer parallelen Fertigung optimal auszunutzen, den Einsatz von Aufbau- und Verbindungstechniken zu beschränken und die Größe des Systems zu minimieren (zu den Möglichkeiten und Vorteilen siehe auch [Riethmüller89]). Andererseits hat eine derart weitgehende Integration auch Nachteile:

- Bei der Herstellung müssen die für die Sensorfertigung und die Elektronikfertigung verwendeten Prozeßschritte miteinander verträglich sein; dies schränkt die Arten der realisierbaren Sensoren ein.
- Zusätzliche Prozeßschritte für Sensoren werden die Ausbeute bei der Produktion verringern.
- Eine monolithische Integration wird in aller Regel Silizium als Grundmaterial verwenden, dies schränkt die Zahl der möglichen Sensoren ebenfalls ein.
- Von außen kommende Signale, die von den Sensoren gemessen werden sollen, können Störungen in der Elektronik verursachen. Eine hohe Integrationsdichte erschwert eine Vermeidung der Störungen durch Abschirmung der Elektronik.
- Einige Sensortypen (chemische, biologische) haben nur eine begrenzte Lebensdauer und müssen auswechselbar sein, was bei fester Integration nicht möglich ist.
- Der für die hohe Integration erforderliche Aufwand bei Design und Herstellung wird erst bei höheren Stückzahlen kompensiert, was Anwendungen mit geringer Stückzahl ausschließt.
- Einzelne Komponenten können nur bedingt unabhängig voneinander entworfen werden.

In [Senturia86] wird geraten, den zusammen mit den Sensoren realisierten Teil der Elektronik zu minimieren. Auch im Bereich der Mikroelektronik findet eine monolithische Integration von digitalen und analogen Funktionen eher selten statt. Eine Alternative zu einer monolithischen Integration ist eine Integration von einzelnen Chips in Multichip-Module [Schuch92] oder ganz allgemein die hybride Integration von mikromechanischen und mikroelektronischen Komponenten auf einem Substrat mit Aufbau- und Verbindungstechniken [Reichl89].

Aus diesen Gründen ist die Höhe des fertigungstechnischen Integrationsgrads nur bedingt ein Kriterium für Mikrosysteme. Die in dieser Arbeit durchgeführte Modellierung geht nicht von der Voraussetzung aus, daß Mikrosysteme weitgehend monolithisch hergestellt werden. Allerdings müssen die durch die Fertigungstechnik gegebenen Randbedingungen, was z.B. Formgebung und kleinste realisierbare Strukturen angeht, in ihren Auswirkungen auf das Systemdesign in der Modellierung berücksichtigt werden.

Größe von Mikrosystemen
Größe kann sich auf verschiedene Eigenschaften beziehen: geometrische Abmessungen, Gewicht, Leistungsaufnahme. Es gibt Anwendungen, z.B. in der Medizintechnik, wo die geometrischen Abmessungen des Systems eine wichtige Rolle spielen (spektakulärstes Beispiel ist sicher die Idee des sich durch Adern im menschlichen Körper bewegenden Mikroroboters), oder in der Raumfahrttechnik, wo eine Reduzierung des Gewicht merkliche Kosten einspart.

Miniaturisierung und geringe Leistungsaufnahme sind auch generell Vorteile der Mikroelektronik. Dennoch haben moderne Prozessoren eine erhebliche Leistungsaufnahme (> 10 Watt) und gehäust einen Flächenbedarf von einigen Quadratzentimetern. Daher können ohne Anwendungsbezug auch für Mikrosysteme Grenzwerte für die Größe, bei deren Überschreitung ein System nicht mehr als Mikrosystem bezeichnet werden kann, nicht angegeben werden. Allerdings wird man erwarten, daß die kleinsten Komponenten bzw. funktionstragenden Einheiten Abmessungen im Bereich von wenigen µm haben werden.

Auf die Modellierung hat die Größe des geplanten Systems, ähnlich wie die eingesetzte Basistechnologie, nur mittelbar Einfluß.

Fazit
Die genannten Kriterien sind nicht unabhängig voneinander, teilweise bedingen sie sich gegenseitig, teilweise schließen sie einander aus. Ihre Anwendung bei der Bewertung von konkreten Systemen unterliegt einem Ermessensspielraum. Manchmal wird auf eine hohe Integration Wert gelegt, manchmal steht die Funktionalität im Vordergrund. Dies hängt von dem geplanten Einsatzgebiet und den daraus folgenden Anforderungen ab. Wenn die Integration einer digitalen Datenverarbeitung als eine notwendige Bedingung für Mikrosysteme betrachtet wird, so ist der in der Einleitung erwähnte Herzkatheter kein Mikrosystem.

Der technologische Fortschritt macht auch eine ständige Neubewertung der Kriterien nötig. Daher fällt es schwer, Mindestanforderungen an Mikrosysteme zu formulieren. Manche Definitionen des Begriffs Mikrosystem fallen oft nur vage aus. Nach [Rauch89] sind Mikrosysteme Systeme, die mehrere integrierte Baugruppen umfassen, von denen mindestens zwei unterschiedliche Funktionen aufweisen, und mindestens eine dieser Funktionen durch nichtelektrische Parameter gekennzeichnet ist.

Es bleibt festzustellen, daß der Begriff Mikrosystem nicht endgültig definiert ist, und daß sich seine Bedeutung mit der Weiterentwicklung der Technologien noch wandeln bzw. präzisieren wird. Trotzdem soll hier schon eine Definition des Begriffs erfolgen: "Mikrosysteme sind unter Anwendung von Mikrotechniken hergestellte Systeme mit Sensoren und/oder Aktoren, die ihre Funktion durch eine intelligente Verknüpfung von nicht notwendigerweise teiloptimierten Komponenten erzielen."

Mikrosysteme sind, etwa im Vergleich zu einzelnen Sensoren, durch eine hohe Komplexität gekennzeichnet. Beim Entwurf auf der Systemebene sind es vor allem die Zahl und Verschiedenartigkeit der Komponenten und der Systemcharakter, die diese Komplexität verursachen. Diese Komplexität stellt eine Hauptschwierigkeit beim Entwurf von Mikrosystemen dar; sie wird in Zukunft noch weiter zunehmen. Daneben ist es die enge Kopplung zur Fertigung, die den Entwurf bestimmt.

2.2 Anforderung an die Modellierungsunterstützung

Aus dieser Analyse von Mikrosystemen lassen sich Anforderungen an die Modellierungsunterstützung ableiten. Diese lassen sich aufteilen in

- Anforderungen an die der Modellierung zugrundeliegende Methode bzw. Methoden.

- Anforderungen an das Werkzeug, mit dem die Modellierung durchgeführt werden soll.

Für eine praktisch durchzuführende Modellierung müssen beide Arten von Anforderungen erfüllt werden, da auch die Anwendung einer geeigneten Methode nur in dem Maß geschehen kann, wie sie durch das Werkzeug unterstützt wird. Die einzelnen Forderungen werden jeweils erst begründet und dann formuliert. Dabei wird zunächst nicht auf Widerspruchsfreiheit geachtet oder darauf, ob und wie die Forderungen in der Praxis umgesetzt werden können.

2.2.1 Anforderungen an die Methoden

Im Unterkapitel 2.1 wurde Komplexität als eine wesentliche Eigenschaft von Mikrosystemen identifiziert. Für das Verständnis komplexer Systeme gibt es bewährte Techniken: Zerlegung, Abstraktion und Hierarchiebildung [Booch91].

Komplexe Systeme können in Teilsysteme oder Komponenten zerlegt werden, die miteinander in Beziehung stehen. Durch Analyse der Komponenten und ihrer Beziehungen kann ein besseres Verständnis des Systems in seiner Gesamtheit gewonnen werden.

Abstraktion bedeutet, von einer komplexen Sache nur die Eigenschaften zu betrachten, die in einem bestimmten Kontext von Interesse sind, und die anderen zu ignorieren. Abstraktion erlaubt es, auch sehr unterschiedliche Dinge miteinander zu vergleichen, indem sie auf einem Abstraktionsniveau beschrieben werden, bei welchem unterschiedliche Eigenschaften ausgeklammert bleiben.[1]

Beide Techniken können rekursiv angewendet werden. Man erhält dann Zerlegungs- und Abstraktionshierarchien. Die Zerlegungshierarchie endet mit Komponenten, die so einfach sind, daß eine weitere Zerlegung keine Verständnisvorteile mehr bringt. Abstraktionshierarchien beginnen mit einer allgemeinen Beschreibung und enden mit einer konkreten Beschreibung von Dingen. Abbildung 2 zeigt je ein einfaches Beispiel für eine Zerlegungs- und Abstraktionshierarchie.

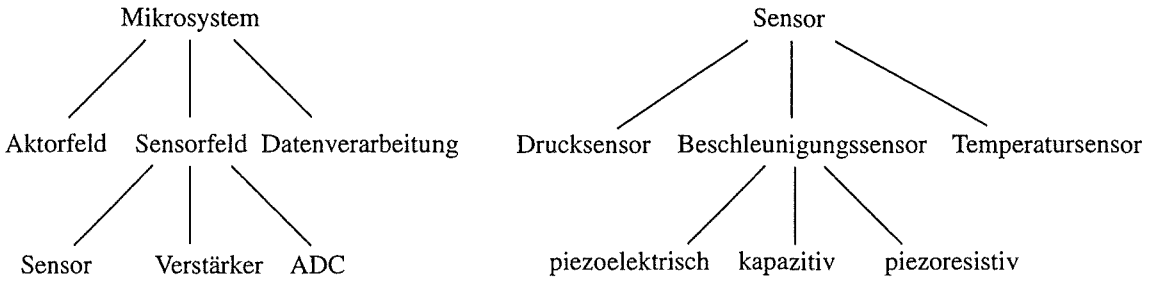


Abb. 2 links: Beispiel für Zerlegungshierarchie, rechts: Beispiel für Abstraktionshierarchie.

Abbildung 1 zeigte bereits implizit die Zerlegung eines Mikrosystems auf verschiedenen Stufen, nämlich als ganzes Mikrosystem, als Bauteilgruppen und als Bauteile. Dies wird in Abbildung 3 veranschaulicht.

---

1. So lassen sich auch Äpfel mit Birnen vergleichen, etwa indem man sie abstrakt als Nahrungsmittel betrachtet, die auf ihren Vitamingehalt hin verglichen werden.

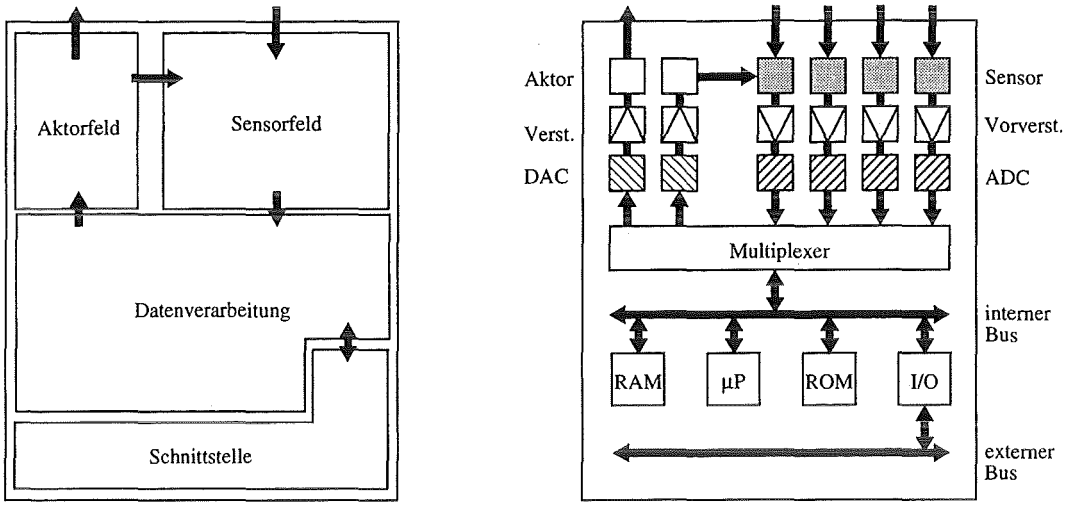


Abb. 3 links: Mikrosystem, zerlegt nach Bauteilgruppen,
rechts: Mikrosystem, zerlegt nach Bauteilen.

Zerlegung und Abstraktion sind nicht eindeutig, sondern hängen vom beabsichtigten Zweck ab. Meist gibt es mehrere mögliche sinnvolle Zerlegungs- und Abstraktionshierarchien. Eine Zerlegung kann sich z.B. an physikalischen Komponenten oder an den realisierten Funktionen orientieren.

Im Bereich der digitalen Mikroelektronik gibt es aufgrund der gesammelten Erfahrungen schon relativ fest definierte Abstraktionsstufen, auf welchen Systeme beschrieben werden können (Transistorebene, Gatterebene, Registerebene etc.). Für Mikrosysteme existiert eine solche feste Einteilung in Hierarchiestufen nicht, da einerseits die Technik noch zu jung ist und andererseits schon bei Komponenten wie den Sensoren die Funktionsmechanismen zu unterschiedlich sind, als daß eine schematische Abstraktion für alle Fälle möglich wäre.

1. Forderung: Die Modellierungsmethode soll die Techniken der hierarchischen Zerlegung und hierarchischen Abstraktion unterstützen.

In der Einleitung wurde dargelegt, daß das Modell statische Merkmale des Mikrosystems abbilden muß. Diese Merkmale variieren sehr stark zwischen den Komponenten des Mikrosystems. Einige Merkmale haben fast alle Komponenten des Mikrosystems gemeinsam (Kosten, Position innerhalb des Systems, Flächenbedarf), andere Merkmale können zwar für viele Komponenten angegeben werden, sind aber nicht für alle sinnvoll (Materialeigenschaften, elektrische Kenngrößen). Wieder andere Merkmale treffen nur für einzelne Komponenten zu (Speicherkapazität, Übertragungsrate). Zwischen verschiedenen Mikrosystemen variieren die beschreibenden Merkmale mit Zahl und Art der verwendeten Komponenten ebenfalls.

2. Forderung: Merkmale eines Systems und seiner Komponenten sollen durch Attribute beschrieben werden können, die in Zahl und Typ variabel sind.

Das dynamische Verhalten des Systems soll ebenfalls modelliert werden können. Dies bezieht sich nicht nur auf das System als Ganzes, sondern auch auf das Verhalten von Komponenten. Eine genaue Komponentensimulation, wie sie beim Komponentenentwurf notwendig ist, soll

nicht in das Modell integriert werden, da sie in vielen Fällen zu aufwendig ist. Vielmehr müssen zur Beschreibung des Komponentenverhaltens vereinfachte Modelle entwickelt werden, welche ausreichend schnell und genau genug sind, um in einer Systemmodellierung eingesetzt zu werden. Solche einfachen Komponentenmodelle werden manchmal auch als Makromodelle bezeichnet [Reichl93]. Die Methode muß die Integration dieser Komponentenmodelle erlauben.

3. Forderung: Das dynamische Verhalten des Systems muß durch die Integration von vereinfachten Komponentenmodellen beschrieben werden können.

Eine weitere Aufgabe des Modells besteht darin, Entwurfsentscheidungen der Entwickler zu beschreiben. Dies ist deswegen notwendig, weil die Gründe für manche Designentscheidung oft nur mit Hintergrundwissen aus einem speziellen Teilgebiet erklärbar sind und daher nicht von allen an der Systementwicklung Beteiligten verstanden werden können. In jedem Mikrosystem wird es auch Parameter geben, die das Gesamtsystem überdurchschnittlich stark beeinflussen, und deren genaue Auswirkungen daher auch für den Entwurf auf Systemebene beschrieben oder erklärt werden müssen. Es ist also wichtig, auch Erfahrungen und Erklärungen in das Modell einbringen zu können.

4. Forderung: Erläuternde Beschreibungen und Erfahrungen der Entwickler des Systems müssen innerhalb des Modells formuliert werden können.

Die Methode sollte von einem potentiellen Anwender leicht erlernt werden können, wenn er sie noch nicht kennt. Der Maßstab dafür, was einfach ist, ist natürlich subjektiv und hängt von der Vorbildung des Modellierers sowie von dem Werkzeug ab, in dem die Methode implementiert ist. Die Erfüllung dieser Forderung soll helfen, Akzeptanzprobleme bei potentiellen Anwendern zu vermindern.

Weiterhin sollte die Methode auch so verständlich sein, daß die mit ihrer Hilfe abgelegten Informationen auch von Anwendern verstanden werden, welche die Methode nur wenig beherrschen. Diese Forderung erscheint nicht zwingend, denn man könnte auch einwenden, daß das Modellierungswerkzeug die abgelegten Informationen auch so präsentieren kann, daß die darunterliegende Methode verdeckt wird und daher keine Rolle mehr spielt. Hier wird aber die Auffassung vertreten, daß die Kommunikation zwischen den an der Entwicklung des Systems Beteiligten durch die völlige Trennung zwischen Methode und Darstellung der Informationen eher erschwert wird.

5. Forderung: Die Methoden sollten einfach und verständlich sein.

Auch die beste Methode besitzt nur theoretischen Wert, wenn sie nicht im Rahmen eines Werkzeugs implementiert ist, mit dem die Modellierung durchgeführt wird. Dabei ist es hilfreich, wenn man bei der Implementierung auf kommerzielle Werkzeuge zurückgreifen kann, welche die Methode unterstützen. Noch besser ist es, wenn die Methode allgemein akzeptiert und verbreitet ist, so daß mehrere kommerzielle Implementierungen zur Verfügung stehen. Dies verringert Zeit und Aufwand für die Realisierung eines Modellierungswerkzeugs. Bei einer noch jungen Technik wie der Mikrosystemtechnik ist dies wichtig, da der Markt noch

nicht das Volumen hat, umfangreiche Eigenentwicklungen zu tragen. In diesem Zusammenhang sei nochmals auf die Verwendung kommerzieller Programmpakete in MEMCAD hingewiesen (siehe Seite 3).

Bei der Implementierung ist auch zu bedenken, daß ein Modellierungswerkzeug nur einen Teilbereich bei der Entwicklung von Mikrosystemen abdeckt. Die Frage nach einer Integration mit anderen Werkzeugen (z.B. für die Entwicklung von Komponenten) muß daher schon bei der Wahl der Methode mit berücksichtigt werden. Je stärker die Unterstützung der Methode durch kommerzielle Werkzeuge ist, desto wahrscheinlicher ist es, daß auch Lösungen für die Integration mit anderen Werkzeugen existieren.

6. Forderung: Die Methode sollte bewährt sein und von kommerziellen Programmen unterstützt werden.

Generell ist zu erwarten, daß für einige Teilbereiche der Modellierung spezielle Methoden besonders gut geeignet sind. Es ist daher zu überlegen, ob eine Modellierung mit vielen, auch speziellen Methoden durchgeführt werden soll, oder nur mit möglichst wenigen, im Idealfall nur mit einer. Die Verwendung mehrerer Methoden hat, besonders im praktischen Einsatz, Nachteile:

- Mehrere Methoden bedeuten einen Mehraufwand bei der Implementierung des Modellierungswerkzeugs.
- Die richtige Integration mehrerer Methoden ist sowohl konzeptionell als auch in der praktischen Umsetzung schwierig.
- Der Lernaufwand steigt stark an. Selbst wenn mehrere Methoden vom Werkzeug unterstützt werden, wird in vielen Fällen der Modellierende nur eine Teilmenge von ihnen sicher beherrschen und für die Modellierung verwenden. Dies führt vor allem dann zu Problemen, wenn mehrere Personen die Modellierung durchführen, die nicht die gleichen Methoden beherrschen.

7. Forderung: Der Modellierung soll eine möglichst kleine Zahl von Methoden zugrundeliegen.

Falls jedoch mit einer Methode allein eine Modellierung nicht erfolgreich durchgeführt werden kann, müssen die eben genannten Nachteile in Kauf genommen werden, und mehrere Methoden verwendet werden. Dann ist wenigstens darauf zu achten, daß diese sich möglichst gut ergänzen und konzeptionell zueinander passen.

8. Forderung: Falls mehrere Methoden verwendet werden, müssen sie integrierbar sein.

2.2.2 Anforderung an das Modellierungswerkzeug

Nachdem oben die Anforderungen an die der Modellierung zugrundeliegenden Methoden aufgestellt wurden, werden jetzt einige Fähigkeiten diskutiert, die das Modellierungswerkzeug selber aufweisen muß. Einige, für praktisch alle Programme geltenden notwendigen Eigenschaften, wie Benutzerfreundlichkeit, Schnelligkeit, Fehlerfreiheit etc. werden nicht im einzelnen aufgeführt. Ebenso ist selbstverständlich, daß das Werkzeug die gewählte Methode ausreichend gut unterstützt.

Eine wichtige Forderung ist die nach geeigneter Darstellung von Informationen. In dem Modell sind viele Daten verschiedener Art abgelegt, die durch eine geeignete Aufbereitung besser zu interpretieren sind. Ein einfaches Beispiel sind Informationen, welche die Plazierung von Bauelementen im System beschreiben, oder Informationen über mikromechanische Strukturen liefern. So könnten Strukturen durch Polygone angenähert sein, deren Daten in dem Modell abgelegt sind. Abbildung 4 zeigt zwei Darstellungsformen derselben Polygondaten und demonstriert den Nutzen einer Visualisierung von Geometriedaten.

(14,14)
(12,16)
(22,16)
(20,14)

Abb. 4 Polygon, repräsentiert durch (x,y)-Eckpunktdaten und durch grafische Skizze. Die Form des Polygons ist der grafischen Darstellung wesentlich leichter zu entnehmen.

Der Umfang der Unterstützung in der Darstellung von Daten kann nicht allgemein angegeben werden. Für Mikrosysteme mit umfangreichen mikromechanischen Komponenten, deren Funktionalität auch durch ihren dreidimensionalen Charakter bestimmt wird, kann die Visualisierung von Geometriedaten durch die Kopplung zu einem kommerziellen CAD-System angemessen sein. Für andere Systeme, bei denen Geometrieinformationen auf der Systemebene kaum relevant sind, reicht möglicherweise eine knappe zweidimensionale Skizze aus, so daß eine einfache Eigenentwicklung vorzuziehen ist. Nichtgeometrische Daten müssen ebenfalls angemessen dargestellt werden können.

Eng mit der Darstellung ist die Aufbereitung von Daten verknüpft. Wenn z.B. in dem Modell Kosten von Komponenten abgelegt sind, sollte es dem Anwender möglich sein, eine nach den Kosten geordnete Aufstellung der Komponenten zu erhalten.

9. Forderung: Die Darstellung und Aufbereitung von Informationen muß in einer der Kommunikation zwischen Entwicklern und Anwendern förderlichen Weise erfolgen.

Falls die im Modell vorhandenen Informationen sehr umfangreich sind, sind Hilfsmittel notwendig, um durch diese Informationen zu "navigieren". Solche Hilfsmittel werden allgemein als Browser bezeichnet; sie beruhen oft auf dem Prinzip, dem Benutzer eine grafische Repräsentation von Informationsstrukturen zu bieten, die leicht zu interpretieren ist. Manchmal werden Informationen auch in einer bestimmten Reihenfolge angezeigt (alphabetisch, chronologisch etc.), um so einen Zugriff auf interessierende Daten zu erleichtern. Die spezielle Auslegung dieser Browser ist von der zur Modellierung verwendeten Methode abhängig.

Ebenso muß ein Benutzer in die Lage versetzt werden, gezielt einzelne Daten im Modell zu verändern. Hierfür sollten spezielle Hilfsmittel zur Verfügung stehen, die aber, wie die Browser, nicht ohne Kenntnis der Methode zu spezifizieren sind.

10. Forderung: Der Benutzer muß beim Zugriff auf das Modell unterstützt werden.

Die Notwendigkeit, für eine dynamische Modellierung vereinfachte Komponentenmodelle zu entwickeln, wurde schon auf Seite 12 (Forderung 3) angesprochen. Die Entwicklung dieser Komponentenmodelle geschieht idealerweise durch den Entwickler der Komponente, da dieser am besten beurteilen kann, wie ein Komponentenverhalten durch eine einfache Modellvorstellung angenähert werden kann. Die Anbindung dieser Komponentenmodelle an das Modell muß möglich sein und auch vom Werkzeug unterstützt werden.

11. Forderung: Komponentenmodelle müssen eingebunden werden können.

Eine Komponente, nämlich die Datenverarbeitung, verdient eine besondere Beachtung bei der Modellierung. Sie wird in vielen Fällen als Software, die auf einem im Mikrosystem vorhandenen Prozessor abläuft, realisiert werden. Diese bestimmt wesentlich das Verhalten und die Fähigkeiten des Mikrosystems und seine Schnittstelle zur Außenwelt. In einem gewissen Sinn bildet sie sogar, ähnliche wie das Modell, eine Abbildung von Eigenschaften des Systems, wenn sie z.B. die Aufgabe hat, eine Korrektur von Sensorfehlern durchzuführen und daher diese Fehler innerhalb der Software "bekannt" sein müssen. Ihre Einbindung als Komponente in das Modell kann als vereinfachte Version erfolgen (analog den anderen Komponentenmodellen). Es kann sich aber auch als zweckmäßig erweisen, die gesamte, für das reale Mikrosystem bestimmte Software, fast unverändert in die Modellierung aufzunehmen. Dies kann auf verschiedene Weisen geschehen; die Diskussion darüber findet sich auf Seite 57.

12. Forderung: Die Software des Systems muß in das Modell einbezogen werden können.

Die Unterstützung der Zusammenarbeit der an der Systementwicklung beteiligten Personen ist eine wichtige Motivation für das Anfertigen eines Modells. Demzufolge muß sie diesen Personen auch zugänglich sein. Da im allgemeinen die Entwicklung nicht an einem Ort erfolgen wird, muß auch das Modell dezentral verfügbar gemacht werden. Dies kann durch verteilte Rechner geschehen, die innerhalb eines Netzwerks verbunden sind. Das Modellierungswerkzeug muß dann in einer solchen verteilten Umgebung ablauffähig sein. Diese Anforderung kann nur durch ein Zusammenwirken von Hardware und Software erfüllt werden.

13. Forderung: Das Modellierungswerkzeug muß dezentral verfügbar sein.

Die Durchführung der Modellierung wird sich über einen längeren Zeitraum, teilweise parallel zur Realisierung des Mikrosystems, erstrecken. Dabei sollen verschiedene Modellvarianten durchgespielt werden können. Auch nach dem Abschluß einer Modellierung kann nie ausgeschlossen werden, daß das Modell nicht für eine Weiterentwicklung des Systems wieder verändert werden muß. Varianten und Weiterentwicklungen müssen so durchgeführt werden können, daß ein einmal erreichter Stand nicht verloren geht. Das Werkzeug muß also auch eine Versionsverwaltung der Modelle durchführen können.

14. Forderung: Von einem Modell müssen mehrere Varianten parallel gehalten werden können, ältere Versionen müssen ebenfalls wieder aktiviert werden können.

Das Modellierungswerkzeug ist nur eines von mehreren Hilfsmitteln für den Entwurf. Komponentenentwurfswerkzeuge, Testumgebungen, Qualitätssicherungssysteme, Systeme zur Unterstützung der Fertigung von Mikrostrukturen [Brauch92] sowie Datenbanksysteme werden in den verschiedenen Stadien des Entwurfs und der Herstellung zum Einsatz kommen. Eine Kopplung dieser Werkzeuge ist für eine gemeinsame Benutzung notwendig. Das Modellierungswerkzeug muß sich daher mit anderen Werkzeugen in eine umfassendere Entwicklungsumgebung integrieren lassen. Dies schließt auch die Einbeziehung vorhandener Grafik- und Sprachmittel zur Unterstützung der Kommunikation der beteiligten Entwickler mit ein.

15. Forderung: Das Modellierungswerkzeug muß mit anderen im Rahmen des Entwurfs verwendeten Werkzeugen integrierbar sein.

Über die Implementierung der Methode hinaus gibt es also eine Reihe von wichtigen Anforderungen an das Modellierungswerkzeug. Eine vollständige Implementierung übersteigt bei weitem das im Rahmen dieser Arbeit Mögliche. Einige dieser Forderungen werden durch prototypische Implementierungen im Rahmen der Modellierung des realen Systems erfüllt, für andere wird ein Weg zur Realisierung aufgezeigt.

2.3 Bewertung von Modellierungsmethoden

Nachfolgend werden einige Methoden zur Modellierung von Systemen vorgestellt und diskutiert. Dabei werden zunächst die Verfahren vorgestellt, die nicht oder nur teilweise geeignet erscheinen, die gestellten Anforderungen zu erfüllen. Dabei ist es im Rahmen dieser Arbeit nicht möglich, jede Methode detailliert darzustellen und ihre Vor- und Nachteile zu erläutern; hier muß auf die einschlägige Fachliteratur verwiesen werden. Es werden die wichtigsten Vorteile genannt, vor allem aber die Nachteile, aufgrund derer die Verfahren verworfen werden. Das ausgewählte Verfahren der objektorientierten Modellierung bedarf natürlich einer eingehenden Darstellung und Bewertung; dies bleibt dem nächsten Kapitel vorbehalten.

Da im Modell Wissen über das zu realisierende Mikrosystem abgelegt ist, stammen einige der betrachteten Methoden aus dem Bereich der KI (Künstliche Intelligenz), der sich mit der Organisation und Darstellung von Wissen[1] beschäftigt. Die dabei entwickelten Verfahren sollen Wissen nicht nur vollständig darstellen, sondern es auch so kodieren, daß aus dem vorhandenen Wissen durch Inferenz (Schlußfolgerung) neues Wissen erzeugt werden kann.

Logik

Mathematische Logik in ihren verschiedenen Ausprägungen (Aussagenlogik, Prädikatenlogik) ist im Bereich der KI sehr weit verbreitet, da altbekannte Methoden zur logischen Inferenz verwendet werden können, neues Wissen zu gewinnen. Für die Modellierung von Mikrosystemen ist sie aber nicht geeignet, weil sie, speziell für Laien, sehr unanschaulich ist (Forderung 5). Dies verdeutlicht ein Beispiel aus [Shapiro87]:

$$1.\ \forall x\ \mathrm{Rad}(x) \rightarrow [\exists y\ \mathrm{Reifen}(y) \wedge \mathrm{umgibt}(y,x)]$$

1. Nach [Tanimoto90] kann Wissen als Menge von Informationen aufgefaßt werden, die so aufgebaut sind, daß sie zum Problemlösen eingesetzt werden können. Wissen besteht aus einer Menge verwandter Fakten, Prozeduren, Modelle und Heuristiken. Es kann sowohl vom Inhalt als auch vom Erscheinungsbild her sehr unterschiedlich sein.

2. $\forall x,y,z$ [umgibt$(x,y) \wedge$ umgibt$(x,z)] \rightarrow y = z$

3. $\forall x,y,z$ [Rad$(y) \wedge$ Rad$(z) \wedge$ Reifen$(x) \wedge$ umgibt$(x,y) \wedge$ umgibt$(x,z)] \rightarrow y = z$

Diese logischen Formeln machen Aussagen über Räder (z.B. von Kraftfahrzeugen) und Reifen. Formel 1 besagt, daß alle Räder von Reifen umgeben sind; Formel 2 stellt fest, daß ein Objekt nur genau ein anderes Objekt umgeben kann; Formel 3 (ableitbar aus Formel 1 und 2) trifft die Feststellung, daß kein Reifen auf mehr als einem Rad sitzen kann.

Neben der Unanschaulichkeit ist die fehlende Möglichkeit der Wissensorganisation ein weiterer Nachteil der Logik. Sollen viele Sachverhalte dargestellt werden, so wird die Wissensbasis schnell unübersichtlich.

Regeln

Regeln bestehen aus Bedingungen und Aktionen. Die meisten Expertensysteme verwenden sie als wesentliche Methode, um Wissen zu repräsentieren. Folgendes Beispiel zeigt eine Regel, die Wissen über Fehlerursachen in Kraftfahrzeugen repräsentiert:

Wenn	der Motor nicht anspringt der Anlasser sich nicht bewegt die Batterie in gutem Zustand ist
Dann	Anlasser ist defekt Anlasser muß ausgetauscht werden

Regeln alleine bilden noch nicht das vollständige Wissen, sie arbeiten zusammen mit einer Faktenbasis. Falls die im Wenn-Teil der Regel aufgestellten Bedingungen in der Faktenbasis vorhanden sind, können die Aktionen im Dann-Teil ausgelöst werden. Man sagt auch, die Regel "feuert". Die Aktionen können auch die Faktenbasis verändern und dadurch das Feuern anderer Regeln bewirken.

Regeln besitzen den Vorteil, daß sie meist eine einfache umgangssprachliche Entsprechung haben (siehe Beispiel). Wissen kann in kleinen Einheiten in die Wissensbasis eingegeben werden; je mehr Regeln dort abgelegt sind, desto mächtiger wird das System. Sie können nicht nur kausale Zusammenhänge, sondern auch heuristisches Wissen, also Erfahrungen, abbilden (Forderung 4). Sie werden durch kommerzielle Werkzeuge, z.B. Expertensystemshells, relativ gut unterstützt.

Neben diesen Vorteilen haben Regeln als Wissensrepräsentationsform auch schwerwiegende Nachteile. Größere Datenmengen lassen sich nur sehr schlecht in ihnen speichern. In [Li91] werden regelbasierte Systeme als unwartbar, untestbar und unzuverlässig bezeichnet. Als wesentliche Ursache dafür werden mangelnde Modularität und fehlende Abstraktionsmöglichkeiten genannt. Für die Beschreibung von komplexen Mikrosystemen sind sie daher nicht geeignet (Forderung 1).

Semantische Netzwerke

Semantische Netzwerke repräsentieren Wissen als durch Relationen verbundene Knoten. Die Knoten entsprechen Gegenständen, Zuständen, Ereignissen etc. in der Realität. Ursprünglich dazu gemacht, den Inhalt natürlichsprachiger Sätze abzubilden, werden sie auch dazu einge-

setzt, um andere Sachverhalte zu modellieren. In [Specht91] werden semantische Netze eingesetzt, um technische Gegenstände in der Konzeptphase des Konstruierens zu beschreiben, noch ehe die geometrischen Dimensionen festgelegt sind.

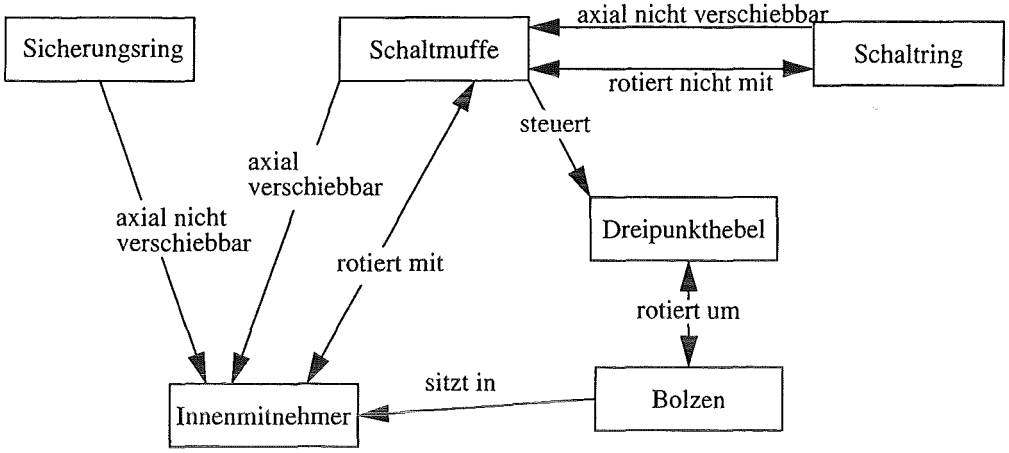


Abb. 5 Semantische Modellierung eines mechanischen Betätigungskomplexes einer mechanischen Reibkupplung [Specht91].

Semantische Netze sind grafisch leicht darzustellen (siehe Abbildung 5) und können Wissen relativ gut strukturieren.

Allerdings gibt es für semantische Netzwerke keinen Standard. Je nach Anwendungsfall und Implementierung unterscheiden sie sich daher in ihren Fähigkeiten. Dynamisches Verhalten kann mit ihnen nicht abgebildet werden (Forderung 3). Wegen des fehlenden Standards ist die Unterstützung durch kommerzielle Werkzeuge schlecht (Forderung 6).

Frames

Frames bieten eine Möglichkeit Wissen, das für ein Objekt oder in einer Situation von Bedeutung ist, zusammenzustellen. Meistens geschieht dies durch eine Liste von Slots (Attributen) und Füllern (Werten). Frames entsprechen somit etwa dem Datentyp "struct" in der Programmiersprache C (bzw. "record" in Pascal). Als Beispiel zeigt Abbildung 6 einen (nicht vollständigen) Frame für ein Mikrosystem.

Attribute	Werte
Aufgabe	Beschleunigungsmessung
Zahl Sensoren	6
Zahl der Aktoren	0
Leistungsaufnahme(Watt)	1.5
Hersteller	KfK

Abb. 6 Beispiel eines Frames für die Beschreibung von Mikrosystemen.

Frames sind gut geeignet, eine Menge gleichartiger Gegenstände oder Situationen zu beschreiben. So kann der Frame im Beispiel ohne Werte als allgemeingültiger Frame verwendet werden, von dem Instanzen gebildet werden, deren Attribute dann mit Werten belegt werden. So können nicht nur ein Student, sondern viele Studenten beschrieben werden. Sie sind relativ leicht zu verstehen und anzuwenden.

Das Framekonzept in seiner einfachen Form ist nicht mächtig genug für die Beschreibung von Mikrosystemen. Es fehlen Möglichkeiten zur Beschreibung von dynamischem Verhalten und zur Hierarchiebildung. In der Praxis findet man oft eine Verbindung zwischen Frames und semantischen Netzen, bei denen Frames an den Knoten des Netzes sitzen. Oft werden Frames auch in Vererbungshierarchien[1] eingebunden und mit prozeduralen Erweiterungen versehen. Der Unterschied zwischen framebasierten Sprachen und objektorientierten Sprachen ist dann nur noch sehr gering [Shapiro87].

Informelle Beschreibungen (Texte, Grafiken, Datenblätter etc.)
Natürlichsprachliche Beschreibungen in textueller Form, ergänzt durch Abbildungen, sind ein geeignetes Mittel, auch komplexe Wissensinhalte auszudrücken[2]. Auf die Entsprechung zwischen Regeln und natürlicher Sprache wurde bereits hingewiesen, auch das gezeigte Beispiel eines semantischen Netzwerks hat einen hohen Anteil von sprachlichen Ausdrücken. Es gibt verschiedene spezielle Ausprägungsformen dieser Darstellungsart, z.B. Datenblätter, mit denen technische Geräte beschrieben werden. Dynamisches Verhalten kann durch die grafische Darstellung funktionaler Zusammenhänge verdeutlicht werden.

Sprache ist eine natürliche Kommunikationsform zwischen Menschen, daher entfällt eine Einarbeitungszeit in die Methode. Weitere Vorteile ergeben sich durch eine geeignete Rechnerunterstützung, hierbei ist die Ablage von Dokumenten in Verwaltungssystemen denkbar, oder eine Aufbereitung der Information im Rahmen eines Hypertextsystems [Kuhlen91], welches die physische Struktur von Texten von der logischen trennt, und die Verbindung konzeptionell zusammenhängender Teile ermöglicht.

Nachteilig bei Verwendung von Sprache in einem rechnergestützten Modell ist, daß das in ihr enthaltene Wissen im wesentlichen nur in der Form angezeigt werden kann, in der es eingegeben wurde. Eine maschinelle Weiterverarbeitung, in Form von Konsistenzüberprüfungen, Test auf Widerspruchsfreiheit, oder Transformation in eine andere Darstellungsform, ist kaum möglich.

Relationen
Eine bewährte Methode für die Modellierung umfangreicher, regelmäßig strukturierter Informationsmengen sind *n*-stellige Relationen. Sie ist deswegen interessant, weil sie die Grundlage für viele kommerzielle Datenbanksysteme ist und daher wichtige Implementierungsanforderungen (Forderungen 6,13,15) erfüllt sind. Relationen werden meist in Tabellenform dargestellt, Abbildung 7 zeigt eine 4-stellige Relation, die Daten von Mikrosystemkomponenten enthält.

Nachteile von Relationen sind, daß komplexe, hierarchische Strukturen nicht oder nur umständlich dargestellt werden können. Ebenso kann dynamisches Verhalten nicht modelliert werden. Bezeichnenderweise findet im Bereich der relationalen Datenbanksysteme beim Da-

1. Der Begriff der Vererbung wird im nächsten Kapitel erklärt.
2. Die vorliegende Arbeit verwendet gerade diese Methode der Wissensdarstellung.

KOMPONENTE			
Kategorie	*Prinzip*	*Funktion*	*ext. Spannung (Volt)*
Sensor	mechanisch	Beschleunigungsmesser	0
Sensor	chemisch	Bestimmung ph-Wert	5
Aktor	fluidisch	Schalter	0
Aktor	elektrisch	Motor	70

Abb. 7 4-stellige Relation KOMPONENTE, dargestellt als Tabelle

tenbankentwurf oft erst eine Modellierung der Daten mit dem Entity-Relationship-Modell (spezielle Art eines semantischen Netzes) statt. Erst danach werden die Daten in Relationenform abgebildet.

Insgesamt erscheinen Relationen als zu eingeschränkt, um die Komplexität eines Mikrosystems zu modellieren. Selbst in einer Komponentenentwurfsumgebung für mikromechanische Strukturen wie MEMCAD wird aus diesen Gründen für die Speicherung von Materialdaten nicht direkt eine relationale Datenbank, sondern das System GESTALT [Heytens89] verwendet, welches dem Anwender objektorientierte Modellierungstechniken zur Verfügung stellt.

Endliche Automaten

Während die meisten bisher besprochenen Verfahren hauptsächlich den statischen Teil einer Modellierung abdecken, sind endliche Automaten (engl. Finite State Machines) dafür geeignet, das Verhalten von Systemen zu beschreiben. Sie sind charakterisiert durch eine Zahl von internen Zuständen, zwischen denen sie aufgrund von Ereignissen wechseln. Automaten lassen sich durch Zustandsdiagramme darstellen (siehe Abbildung 8). Komplexere Geräte wie

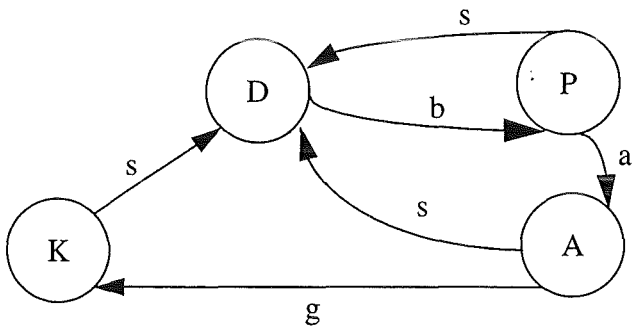


Abb. 8 Zustandsdiagramm eines endlichen Automaten, der einen Teil einer Mikrosystemsteuerung beschreibt. Der Automat ist in einem der Zustände K (keine Daten vorhanden), D (Datenaufzeichnung läuft), P (Datenpuffer belegt), A (Ausgabe läuft) und wechselt zwischen ihnen bei den Ereignissen s (Messung starten), b (Messung beendet), a (Daten ausgeben), g (Datenpuffer geleert).

Computer sind durch eine hohe Zahl von möglichen Zuständen gekennzeichnet. Geht man davon aus, daß der Speicherinhalt den Zustand eines Computers definiert, so erhält man bei einer Speichergröße von 512 KByte ca. $10^{1300000}$ mögliche interne Zustände [Rembold87].

Es kommt also bei der Modellierung mit endlichen Automaten entscheidend darauf an, Zustände geeignet zu definieren, damit ihre Zahl sich in beherrschbaren Grenzen hält. Unterstützt wird dies durch eine Erweiterung der einfachen Zustandsdiagramme zu Statecharts, welche die Definition von hierarchischen und parallelen Zuständen erlauben [Harel87].

Vorteilhaft erscheint bei Automaten, daß sie mit Zustandsdiagrammen oder Statecharts eine Darstellungsform haben, welche es erlaubt, das mit ihnen modellierte dynamische Verhalten grafisch wiederzugeben. Dies erhöht die Verständlichkeit.

Während endliche Automaten geeignet erscheinen, die Steuerung von Mikrosystemen zu beschreiben, sind sie zur Beschreibung von statischen Eigenschaften nicht geeignet. Ebenso werden physikalische Vorgänge, die durch eine zeitkontinuierliche Veränderung von Größen gekennzeichnet sind, von ihnen nicht adäquat dargestellt.

Prozedurale Programmiersprachen
Viele physikalische Vorgänge in Mikrosystemkomponenten lassen sich durch Differentialgleichungen und ihre Lösungen geeignet beschreiben. Allerdings kann diese Methode sehr rechenintensiv sein, da für komplexe Systeme die Zahl der Gleichungen hoch ist und ihre Lösungen meist nur auf numerischem Weg zu erhalten sind. Für eine Beschreibung auf Systemebene wird man daher versuchen, Differentialgleichungen für idealisierte Komponenten aufzustellen, die nur die Abhängigkeiten weniger, charakteristischer Größen beschreiben, und eine geschlossene analytische Lösung haben. Diese Lösung läßt sich dann durch Darstellung in einer Programmiersprache wie C, Pascal oder Fortran in eine rechnertaugliche Form überführen.

Nachteilig ist, daß in dieser Form das der Differentialgleichung zugrundeliegende idealisierte Modell der Komponente oder des physikalischen Vorgangs nicht mehr explizit vorhanden ist. Ebenso sind Überlegungen des Entwicklers, welche Aspekte der Wirklichkeit im idealisierten Modell nicht wiedergegeben werden und warum sie vernachlässigt werden können, nicht mehr nachzuvollziehen.

Fazit
In den vorangehenden Abschnitten wurden verschiedene Methoden zur Wissensrepräsentation mit spezifischen Vor- und Nachteilen präsentiert. Oftmals wird eine Kombination von mehreren Methoden eingesetzt in der Hoffnung, die Vorteile der einzelnen Verfahren zu kombinieren. Eine übliche Kombination in Expertensystemshells ist die aus Frames und Regeln, wobei mit Frames der statische Teil eines Wissensgebiets modelliert wird und mit Regeln das prozedurale und heuristische Wissen abgebildet wird. Die Probleme, die sich aus einer solchen Kombination mehrerer Methoden ergeben können, wurden bei Begründung von Forderung 7 genannt.

Die bisher genannten Methoden erfüllen die gestellten Forderungen jeweils nur zum Teil. Im nächsten Kapitel wird deshalb eine weitere Methode ausführlich vorgestellt, welche die meisten Forderungen erfüllt.

3 Methode und Implementierung

Nachdem verschiedene Modellierungsverfahren diskutiert und als unzureichend erkannt wurden, soll jetzt die Methode der objektorientierten Modellierung beschrieben und bewertet werden. Danach wird dieser Ansatz anhand der im letzten Kapitel aufgestellten Forderungen überprüft und Defizite durch die Integration von informellen Beschreibungen ausgeglichen. Für die prototypische Implementierung eines Modellierungswerkzeugs wird ein kommerzielles Werkzeug ausgewählt und durch Eigenentwicklungen geeignet ergänzt.

3.1 Methode der objektorientierten Modellierung

3.1.1 Grundkonzepte des objektorientierten Paradigmas

Objektorientierung (OO) ist eines, wenn nicht das bestimmende Paradigma in der Informatik zu Beginn der 90er Jahre. Programmiersprachen, CASE, Datenbanken, Betriebssysteme, KI, vernetzte Systeme sind Bereiche, in denen Neuentwicklungen oder Erweiterungen auf der Grundlage der Objektorientierung stattfinden. Historisch lassen sich im wesentlichen 3 Bereiche ausmachen, die Beiträge zu diesem Gebiet geleistet haben: Programmiersprachen, Datenbanksysteme und KI.

Im Bereich der Programmiersprachen gilt das Ende der 60er Jahre entwickelte SIMULA als erste Sprache, die wesentliche objektorientierte Elemente einsetzt[1]. Dennoch dauerte es ungefähr 20 Jahre, bis mit C++ (als Erweiterung der Sprache C) eine OO-Sprache entwickelt wurde, die sich auch im kommerziellen Bereich behaupten kann. Inzwischen gibt es eine Reihe von OO-Sprachen, die entweder Neuentwicklungen darstellen wie Smalltalk und Eiffel, oder aus älteren Sprachen durch die Hinzunahme von OO-Elementen entstanden sind. Neben C++ sind hier Object Pascal, CLOS (Common Lisp Object System) oder Objective-C zu nennen, eine objektorientierte Variante von Ada ist in Vorbereitung. Obwohl die Verbreitung von OO-Sprachen nur abgeschätzt werden kann, ist unstrittig, daß C++ die bei weitem am meisten verwendete OO-Sprache ist. Dies bewirkt auch, daß Implementierungen und Werkzeuge für die Programmentwicklung in C++ auf den Rechnern vieler Hersteller verfügbar sind.

Im Bereich der Datenbanksysteme haben die Beschränkungen von relationalen Systemen in der Datenmodellierung zur Entwicklung von semantischen Datenmodellen geführt, welche die Zuordnung von Attributwerten zu Objekten und die Darstellung von Beziehungen zwischen verschiedenen Objekten ermöglichen [Heuer92]. Das bekannteste von ihnen ist das sogenannte Entity-Relationship-Modell (ER-Modell) [Chen76]. Sie werden aber meist nur als Entwurfswerkzeuge für relationale Datenbanken eingesetzt. Auf diesen aufbauend wurden dann objektorientierte Datenmodelle entwickelt, die als Grundlage für objektorientierte Datenbanksysteme (OODBS) dienen. Inzwischen sind eine Reihe von OODBS kommerziell erhältlich.

Im Bereich der KI sind es im wesentlichen die in Kapitel 2 erwähnten semantischen Netze und Frames, die als Vorstufen einer objektorientierten Wissensrepräsentation gesehen werden können. Der Übergang dabei ist fließend, eine fortgeschrittene framebasierte Sprache ist einer

---

1. Das bedeutet nicht, daß die Sprache heute gänzlich veraltet ist; in [Meyer90] wird sogar die Auffassung vertreten, daß SIMULA eine Verbesserung gegenüber den meisten ihrer Nachfolger ist, da die meisten Konzepte schon in ihr richtig eingeführt werden.

objektorientierten Sprache sehr ähnlich [Shapiro87] (wobei eine objektorientierte Sprache mehr als praktisch einzusetzende Programmiersprache angesehen wird, eine framebasierte Sprache eher ein Hilfsmittel der KI ist). Als eine Gruppe von Werkzeugen, die diese Konzepte unterstützen, sind Expertensystemshells zu nennen.

Obwohl die eingangs dieses Unterkapitels genannten Bereiche, in denen OO-Techniken eingesetzt werden, recht heterogen sind, lassen sich im wesentlichen doch zwei Punkte nennen, die diesen Bereichen gemein sind und mit denen der Einsatz von OO-Techniken motiviert wird. Zum einen ist es der Wunsch, Problemlösungen auf dem Rechner nicht auf einer niedrigen rechnernahen Ebene, sondern in Begriffen der Anwendung zu formulieren. Zum anderen versucht man, die bei der Bearbeitung größerer Probleme auftretende Komplexität durch geeignete Beschreibungsverfahren zu beherrschen[1].

Bevor jetzt die Methode der objektorientierten Modellierung näher erläutert wird, muß darauf hingewiesen werden, daß im Bereich der Objektorientierung eine Vielzahl von Begriffen nicht eindeutig definiert ist, und daß sogar sich widersprechende Definitionen existieren. Neben der unbestimmten Terminologie gibt es miteinander konkurrierende Konzepte und Methoden. Dies hat mehrere Ursachen:

- Die bereits erwähnte Breite der Anwendungsbereiche (CASE, Datenbanken, Betriebssysteme etc.) macht eine für alle Bereiche gültige Definition von Begriffen beinahe unmöglich.
- Ebenso hat die historische Entwicklung von OO in den drei Teilgebieten Sprachen, Datenbanksysteme und KI Begriffe unterschiedlich geprägt.
- OO ist ein sehr aktuelles Forschungsgebiet, so daß permanent neue Konzepte und Denkansätze entwickelt werden, Standards dagegen nur ansatzweise existieren. Im Bereich der OO-Programmierung weisen die verschiedenen Sprachen teilweise sehr unterschiedliche Eigenschaften auf, die kontrovers diskutiert werden. Bei den Datenbanksystemen gibt es das sogenannte Manifesto [Atkinson89], welches notwendige Eigenschaften von OO-Datenbanksystemen beschreibt, welches aber nicht als endgültig und verbindlich angesehen wird.[2]

Die folgenden Ausführungen beschränken sich daher auf Konzepte, die vielen (aber nicht allen) OO-Ansätzen zugrunde liegen. Strittige Konzepte können im Rahmen dieser Arbeit nur sehr eingeschränkt diskutiert werden. Viele Überlegungen und Definitionen sind an [Booch91] angelehnt, der seinerseits aus einer umfangreichen Literatur zitiert.

Der zentrale Begriff der OO-Modellierung ist der des *Objekts*. Ein Objekt modelliert einen Teil der Wirklichkeit. Dabei kann es sich um physikalische Gegenstände handeln oder immaterielle Dinge wie Prozesse oder Ereignisse. Objekte sind gekennzeichnet durch Zustand, Verhalten und Identität.

---

1. Im Bereich der Programmiersprachen gibt es noch die zusätzliche Motivation, durch den Einsatz objektorientierter Methoden und Sprachen wiederverwendbare Software zu schreiben.
2. Mit C++ scheint sich ein de-facto Standard bei den Programmiersprachen zu etablieren, welcher auch OO-Datenbanksysteme maßgeblich beeinflußt.

Zustand bezeichnet die Gesamtheit der statischen Eigenschaften des Objekts (diese Eigenschaften werden als *Attribute* bezeichnet) und der (meist veränderbaren) Werte der Attribute. Komplexere Objekte werden nicht nur durch Attribute beschrieben, sie können vielmehr ihrerseits Unterobjekte enthalten.

Verhalten ist gekennzeichnet durch die Art, wie ein Objekt agiert und reagiert, d.h. welche Aktionen es veranlaßt oder selbst ausführen kann. Diese Aktionen sind oft mit einer Änderung seines Zustands verbunden; sie werden auch als *Methoden* des Objekts bezeichnet, die dann ausgeführt werden, wenn das Objekt eine entsprechende *Botschaft* eines anderen Objekts[1] erhält. Diese Methoden werden in aller Regel in einer prozeduralen Programmiersprache implementiert.[2]

Die Identität eines Objekts unterscheidet es von allen anderen Objekten, d.h. selbst Objekte, welche in Zustand und Verhalten gleich sind, können mittels ihrer Identität unterschieden werden.

Durch die Definition eines Objekts wird das Modell geteilt in einen Bereich, der außerhalb des Objekts liegt, und einen Bereich, der innerhalb des Objekts liegt. Je nach Grad der *Einkapselung* ist der im Innern liegende Bereich des Objekts gegenüber Eingriffen von außen geschützt. Dieses Konzept wird sehr uneinheitlich gehandhabt: manchmal fehlt eine Einkapselung völlig, manchmal sind die Attribute des Objekts gekapselt, die Methoden aber nicht, und manchmal kann der Grad der Kapselung wie in C++ individuell für Attribute und Methoden eingestellt werden.

Eng mit dem Begriff Objekt verbunden ist der der *Klasse*. Eine Klasse beschreibt eine Menge von Objekten, welche eine gemeinsame Struktur und ein gemeinsames Verhalten aufweisen. Struktur bedeutet in diesem Zusammenhang, daß eine Klasse die gleichen Attribute aufweist wie die von ihr beschriebenen Objekte, diesen Attributen aber nicht in jedem Fall Werte zugeordnet sind. Ein einzelnes Objekt wird auch als *Instanz* einer Klasse bezeichnet. Während Objekte einen konkreten Teil der Realität modellieren, repräsentieren Klassen eine bestimmte Abstraktion.[3]

Ein weiteres wichtiges Konzept ist die Technik der *Vererbung* von Attributen und Methoden. Sie ermöglicht es, Klassen als Erben anderer Klassen festzulegen. Diese abgeleiteten Klassen oder *Unterklassen* übernehmen ganz oder teilweise die Struktur und die Aktionen der *Oberklasse* und definieren zusätzlich noch eigene Attribute und Methoden. Hat eine Klasse mehrere Oberklassen, so spricht man von *mehrfacher Vererbung*. In diesem Fall übernimmt die Unterklasse die Strukturen und Aktionen aller Oberklassen. Vererbung kann auch in mehreren Stufen geschehen, d.h. von einer Unterklasse können wiederum andere Klassen abgeleitet werden. Auf diese Weise entstehen Vererbungshierarchien.

---

1. Das Austauschen von Botschaften zwischen Objekten scheint zu implizieren, daß beide Objekte eigenständig sind und parallel arbeiten können. Dies ist in den vorhandenen Implementierungen meist nicht der Fall; die Semantik des Sendens einer Botschaft ist dort vergleichbar mit der des Aufruf eines Unterprogramms in einer prozeduralen Programmiersprache wie C oder Fortran.
2. Eine Alternative wird in der Diskussion am Ende des Kapitels beschrieben.
3. Manchmal werden Klassen auch als "Objektfabriken", die Objekte gleichen Typs "produzieren" können, betrachtet [Atkinson89].

Zwischen Objekten und Klassen bestehen verschiedene Arten von Beziehungen, die schon angedeutet wurden. Zwischen einem Objekt und der Klasse, von der es abgeleitet ist, besteht eine *Instanz-von* Beziehung. Falls ein Objekt selbst wieder Objekte enthält, so besteht zwischen ihm und seinen Unterobjekten eine *hat-ein* (engl. has-a) Beziehung. Zwischen einer Klasse und ihrer Oberklasse besteht eine *ist-ein* (engl. is-a oder ISA) Beziehung, durch die in vielen Fällen eine Spezialisierung ausgedrückt wird.

An einem einfachen Beispiel sollen einige dieser Begriffe veranschaulicht werden. Abbildung 9 zeigt eine einfache Klassenhierarchie für Komponenten, wie sie in Mikrosystemen

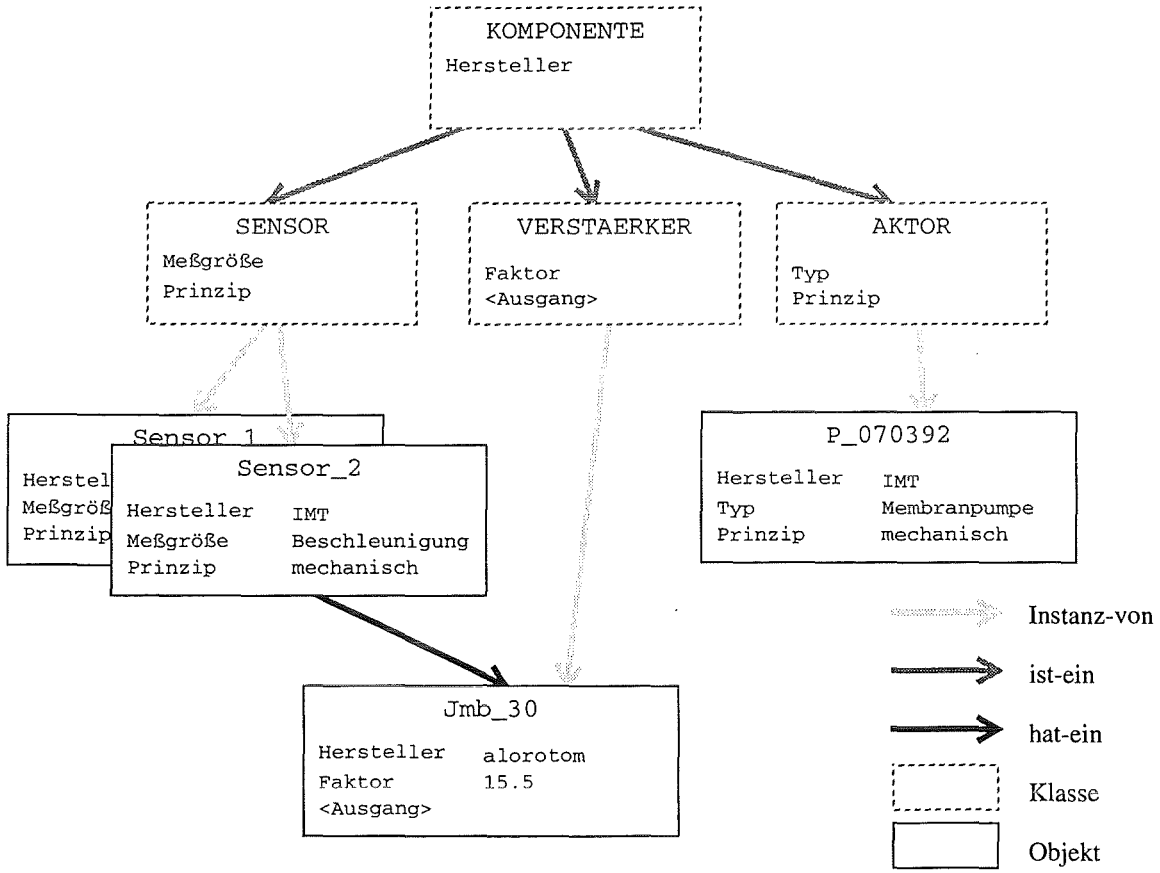


Abb. 9 Einfache Klassen- und Objekthierarchie für Komponenten eines Mikrosystems (nicht vollständig). Bei den Klassen werden nur die jeweils neu hinzukommenden Attribute aufgeführt. Methoden sind durch <> gekennzeichnet.

vorkommen. Jede Komponente hat einen Hersteller, daher wird auf der obersten Abstraktionsstufe eine Klasse KOMPONENTE mit diesem Attribut eingeführt. Von dieser Klasse werden die drei Unterklassen SENSOR, VERSTAERKER und AKTOR abgeleitet, die jeweils eigene Attribute einführen.

Von diesen Klassen sind einige Instanzen abgeleitet, die konkrete Bauteile beschreiben. Die Klasse KOMPONENTE hat keine Objekte, da sie ein Abstraktionsniveau beschreibt, auf dem die Bildung von konkreten Instanzen nicht sinnvoll ist. Alle anderen Klassen haben Objekte, welche alle Attribute der jeweiligen Klassen aufweisen, von denen sie direkt oder indirekt ab-

geleitet sind. Diese Attribute haben bei den Objekten Werte (hier Zahlen oder Strings). Zwischen `Sensor_2` und `Jmb_30` besteht eine hat-ein Beziehung. Sie zeigt hier an, welcher konkrete Verstärker an den Sensor angeschlossen ist.

Der dynamische Aspekt einer Modellierung, der sich in den den Klassen bzw. Objekten zugeordneten Methoden ausdrückt, läßt sich nicht direkt veranschaulichen. In dem Beispiel hat die Klasse `VERSTAERKER` eine Methode `<Ausgang>`, welche für Instanzen dieser Klasse die jeweilige Ausgangsspannung berechnet (dabei wird der aktuelle Wert von `Faktor` verwendet). Hinter einer solchen Methode verbirgt sich in der Regel ein Algorithmus, welcher in einer Programmiersprache wie C++ implementiert ist.

3.1.2 Bewertung des objektorientierten Ansatzes

Im folgenden soll der objektorientierte Ansatz anhand der im letzten Kapitel auf Seite 11 ff. aufgestellten Forderungen an die Modellierungsmethoden auf seine Eignung überprüft werden.

Forderung 1 nach hierarchischer Zerlegung und Abstraktion ist erfüllt. Das erstere läßt sich mit Objekten und Unterobjekten darstellen, das letztere wird durch Klassenhierarchien realisiert. Allerdings zeigt schon Abbildung 9, daß der Begriff Zerlegung die Semantik der Beziehung zwischen Objekt und Unterobjekt nicht immer richtig wiedergibt bzw. die hat-ein Beziehung zwischen Objekt und Unterobjekt unterschiedlich interpretiert werden kann. In der Aussage "Ein Mikrosystem hat ein Sensorfeld" (Abbildung 2) hat die hat-ein Beziehung eine andere Bedeutung als in "Ein Sensor hat einen Verstärker". Im ersten Fall wird ausgedrückt, daß ein Sensorfeld ein Bestandteil eines Mikrosystems ist. Im zweiten Fall ist der Verstärker kein Bestandteil des eigentlichen Sensors, sondern diesem nur durch eine Signalverbindung zugeordnet. Die Semantik der Beziehung zwischen Objekt und Unterobjekt wird auf Seite 30 ff. genauer behandelt.

Die variable Beschreibung von Systemmerkmalen (Forderung 2) ist erfüllt durch die Möglichkeit, in Klassen Attribute zu definieren, die an die Objekte weitergegeben und dort mit Werten besetzt werden. Diese Attribute können einfache Zahlen oder auch komplexere Strukturen wie Felder oder Listen sein.

Die Integration von vereinfachten Komponentenmodellen (Forderung 3) ist erfüllt, da Komponenten als Objekte modelliert werden können, deren Verhalten durch Methoden beschrieben wird. Dies setzt natürlich voraus, daß die einfachen Komponentenmodelle auch in einer Form vorliegen, die eine Integration als Methode möglich macht. Dies wird dann der Fall sein, falls ein Komponentenmodell in einer Programmiersprache wie C++ oder C implementiert ist. Falls das Komponentenmodell selbst nur innerhalb eines anderen Werkzeugs implementiert wurde, läuft die Integration in das OO-Modell auf eine Kopplung von Werkzeugen hinaus. Hier sind generelle Aussagen nicht möglich (siehe dazu auch die Diskussion am Ende des Kapitels).

Erläuternde Beschreibung und Erfahrungen der Entwickler (Forderung 4) können nicht adäquat formuliert werden. Zwar drückt sich in der Wahl der Klassenhierarchie auch Erfahrungswissen des Entwicklers aus, aber wichtige Designentscheidungen und ihre Begründungen können so nicht wiedergegeben werden. Hier muß das Verfahren erweitert oder ergänzt werden. Von den besprochenen Wissensrepräsentationsformen bieten sich informelle Beschrei-

bungen und Regeln zur Modellierung dieser Art von Informationen an. Die Frage einer möglichen Integration eines dieser Verfahren in die OO-Modellierung wird noch diskutiert werden.

Die Verständlichkeit der Methode (Forderung 5) ist von ihrer Implementierung und der Vorbildung des Anwenders abhängig. Die eingesetzten Techniken der Abstraktion und der Zerlegung sind generelle Strategien für die Lösung komplexer Probleme, dennoch ist ihre Anwendung im Rahmen eines objektorientierten Modells nicht eindeutig oder trivial (siehe dazu den nächsten Abschnitt 3.1.3). Andererseits erleichtert die mögliche grafische Repräsentation von Klassen- und Objekthierarchien das intuitive Verstehen, insofern scheint sie zumindest für Betrachter des Modells geeignet.

Forderung 6 nach Unterstützung durch kommerzielle Werkzeuge ist erfüllt, da in den Bereichen Programmiersprachen, Datenbanksystemen und KI Programmpakete zur Verfügung stehen, die als Ausgangspunkt einer Modellierung dienen können. Wie allerdings im Unterkapitel über die Implementierung noch zu sehen ist, können diese nicht ohne Modifikationen übernommen werden. Vorteilhaft ist in diesem Zusammenhang, daß OO-Techniken allgemein eine wichtige Rolle bei der Lösung von Problemen in vielen Bereichen eingeräumt wird, so daß die Unterstützung durch kommerzielle Werkzeuge auch für die absehbare Zukunft gesichert scheint. Nachteilig ist in jedem Fall die schon angesprochene fehlende Standardisierung, was sowohl einige Konzepte wie auch Werkzeuge betrifft.

Forderung 7 nach geringer Zahl von zugrundeliegenden Methoden ist insofern erfüllt, als die objektorientierte Modellierung die meisten Forderungen abdeckt und nur die Repräsentation von Erfahrungswissen und weiterführenden Erklärungen nicht möglich ist. Es ist allerdings noch zu prüfen, ob die Forderung nach einer möglichst kleinen Zahl von Verfahren dann aufgegeben werden sollte, wenn die Integration eines anderen Verfahren leicht möglich erscheint (Forderung 8) und so große Vorteile verspricht, daß die Nachteile (erhöhter Implementierungsaufwand etc.) aufgewogen werden.

Die Methode der objektorientierten Modellierung erfüllt also die wesentlichen formulierten Forderungen. Bevor jetzt ein Weg gezeigt wird, auch Forderung 4 zu erfüllen, wird das prinzipielle Vorgehen bei der objektorientierten Modellierung erläutert.

3.1.3 Vorgehen bei der Methode und kritische Aspekte

Bei der objektorientierten Modellierung kommt es entscheidend darauf an, Objekte richtig zu definieren und geeignete Abstraktionen für die Klassendefinitionen zu finden. Das Vorgehen kann nur prinzipiell und nicht detailliert angegeben werden, da sich Mikrosysteme dazu zu sehr unterscheiden und daher die Anforderungen an das Modell zu unterschiedlich sind.

Finden von Objekten

Objekte lassen sich auf verschiedene Arten finden. Mögliche Kandidaten sind materielle Dinge wie Sensoren, Aktoren, Elektronik usw. Ebenso kann es zweckmäßig sein, nicht die physikalischen Bauteile, sondern die durch sie realisierten Funktionen als Objekte zu modellieren. Eine Methode besteht darin, die Problemstellung durch einen Text zu beschreiben und in diesem Text die Substantive daraufhin zu prüfen, ob sie als Objekte in Frage kommen.

Wichtig ist in jedem Fall, Objekte auf einer geeigneten Abstraktionsstufe zu definieren. Dies muß natürlich für jeden Einzelfall neu entschieden werden. Bei der Beschreibung eines Mikrosystems auf Systemebene ist es nicht sinnvoll, einzelne Transistoren der Elektronik als Objekte der Modellierung zu wählen.

Beim Entwurf von Mikrosystemen wird das Finden von Objekten auch durch die Art des Entwurfs beeinflußt. Falls einzelne Komponenten schon existieren, bieten sie sich als Ausgangspunkt der Objektsuche an. Ist hingegen nur die gewünschte Funktion des Systems bekannt, wird man sinnvollerweise eine funktionale Zerlegung versuchen, und Unterfunktionen als Objekte modellieren.

Finden von Klassen
Wegen der Instanzbeziehung zwischen Objekten und Klassen muß auch die Suche nach ihnen gemeinsam stattfinden. Das Definieren einer Klassenhierarchie ist im wesentlichen ein Klassifikationsproblem: das Erkennen von Gemeinsamkeiten zwischen Objekten führt dazu, daß sie auf einer abstrakten Ebene gemeinsam beschrieben werden können. Leider gibt es keine "richtige" Klassifizierung eines Problems, sie ist vielmehr abhängig von der hinter der Klassifizierung steckenden Absicht, und von demjenigen, der sie durchführt. In Abbildung 9 war die Klassifizierung der Komponenten `Sensor_1`, `Sensor_2`, `Jmb_30` und `P_070392` durch ihre Funktion innerhalb eines Mikrosystems gegeben. Eine andere Einteilung ergäbe sich, wenn die zur Herstellung der Komponente verwendete Technologie (Mikromechanik, Mikroelektronik etc.) als wesentliches Kriterium verwendet würde.

Manchmal ist es auch die Ähnlichkeit mit einem Konzept oder Prototypen, die zu einer Klassifizierung führen kann. So ist es möglich, z.B. eine Klasse für Sensoren zu definieren, um sie von anderen Komponenten abzugrenzen, obwohl es schwer fällt, eine allen Arten von Sensoren gemeinsame Eigenschaft zu finden (bis auf die, daß sie eine Meßgröße in eine andere Größe umwandeln).

Eine gute Klassifizierung ist auch abhängig von der in einem Gebiet vorliegenden Erfahrung. Leider ist diese auf dem Gebiet der Mikrosystemtechnik noch nicht sehr ausgeprägt. Ein Beispiel für die Klassifizierung von Sensoren in der Mikrosystemtechnik gibt [Bortolazzi93b]; als Kategorien für Sensoren werden dort genannt Primärsignal, allgemeine Eigenschaften, Detektionsverfahren, Umsetzungsverfahren und sensitives Material.

Definieren von Attributen
Nach der Wahl der Objekte bzw. Klassen muß festgelegt werden, welche Eigenschaften der realen Gegenstände durch Attribute modelliert werden sollen. Auch hier können keine festen Regeln angegeben werden, die Wichtigkeit von Attributen ist stark kontextabhängig. Eine in der Mikrosystemtechnik sicherlich unerhebliche Eigenschaft ist die Farbe von Gegenständen, also wird sie im Modell nicht auftauchen. Ebenso sollten alle Eigenschaften, die nur auf der Komponentenentwurfsebene wichtig sind, in der Systemmodellierung nicht vorkommen.

Insgesamt ist zu sagen, daß sowohl die Definition von Objekten bzw. Klassen als auch die Festlegung von Attributen eine anspruchsvolle Aufgabe ist, für die bestenfalls Faustregeln, aber keine Patentrezepte angegeben werden können. Manchmal können Objekte gefunden werden, manchmal müssen sie erfunden werden. Jedes Modell gibt nur einen Teil der Wirk-

lichkeit wieder; die Entscheidung darüber, was Bestandteil des Modells sein soll, ist dem Anwender vorbehalten. Weitere Überlegungen dazu werden im nächsten Kapitel bei der Modellierung des Mikrosystems zur Messung von Beschleunigungen angegeben.

Beziehungen zwischen Klassen und Objekten
Klassen und Objekte existieren in einem Modell nicht isoliert voneinander, sondern sind durch Beziehungen miteinander verbunden. Einige wesentliche Beziehungen (ist-ein, hat-ein, Instanz-von) wurden schon erwähnt. Die Semantik dieser Beziehungen ist aber nicht eindeutig, speziell die Beziehung zwischen Unter- und Oberklasse und die Beziehung zwischen Objekt und seinen Unterobjekten kann zur Modellierung mehrerer Beziehungen in der Realität verwendet werden.

Ist-ein Beziehung zwischen Klassen
Die Beziehung zwischen Unter- und Oberklasse wird mit ist-ein bezeichnet, und drückt meistens eine Spezialisierung aus. Die Unterklasse erbt alle Attribute und Methoden der Oberklasse, und fügt eigene hinzu. Faßt man die Oberklasse als Menge der von ihr und ihren Unterklassen gebildeten Instanzen auf, so können Objekte der Unterklasse als Teilmenge der Oberklasse angesehen werden. Diese Interpretation kann aber selbst in scheinbar einfachen Fällen zu Schwierigkeiten führen.

Als Beispiel mögen einfache geometrische Figuren dienen, Rechtecke und Quadrate. In der Klasse für Rechtecke seien als Attribute die beiden Seitenlängen definiert und als Methoden die Berechnung des Flächeninhaltes sowie die Skalierung der Länge einer Seite um einen Faktor. Wenn man nun versucht, Quadrate als Teilmenge der Rechtecke aufzufassen, und die Klasse für Quadrate von Rechtecken abzuleiten, ergeben sich Probleme. Die Modellierung von Quadraten ist nicht optimal, weil beide Seitenlängenattribute vorhanden sind, Quadrate aber durch eine Seite ausreichend beschrieben werden. Die Berechnung des Flächeninhalts erfolgt nach wie vor richtig, aber nicht so effizient wie möglich, weil weiterhin auf zwei Attribute zugegriffen wird. Die Skalierung der Länge einer Seite dagegen ist keine erlaubte Operation mehr, weil sie aus einem Quadrat ein Rechteck machen würde. Die Annahme, Quadrate können als Teilmenge von Rechtecken betrachtet werden, kann also nicht aufrechterhalten werden, weil sie nicht alle für Rechtecke definierbaren Operationen ausführen können[1].

Die Umkehrung der Ableitungsbeziehung, also Rechtecke als Unterklasse von Quadraten, erscheint dagegen möglich. Die Klasse RECHTECK müßte ein zusätzliches Attribut für die Seitenlänge und die Skalierung der Seitenlänge als neue Methode einführen. Die Methode zur Berechnung des Flächeninhalts müßte modifiziert werden, die Semantik der Methode bliebe aber erhalten. Die Unterklasse ist in diesem Fall aber keine Spezialisierung, sondern eine Generalisierung der Oberklasse (was eine Umkehrung der "normalen" Verhältnisse ist).

Noch eine weitere Möglichkeit, diese Klassenbeziehung zu definieren, läßt sich finden, nämlich wenn beide Klassen von einer gemeinsamen Oberklasse abgeleitet werden, die nur eine Methode zur Berechnung des Flächeninhalts definiert, die dann vererbt und in den Unterklassen entsprechend modifiziert wird. Abbildung 10 zeigt die drei Klassenbeziehungen.

---

1. Bei einigen, meist aus der KI stammenden Modellierungswerkzeugen, versucht man das Problem dadurch zu lösen, daß die Vererbarkeit von Attributen und Methoden für jede Klasse individuell gesteuert werden kann (ein Vorschlag dazu findet sich in [Lee93]). Die Semantik der Ober/Unterklassebeziehung wird aber dadurch noch unkalkulierbarer.

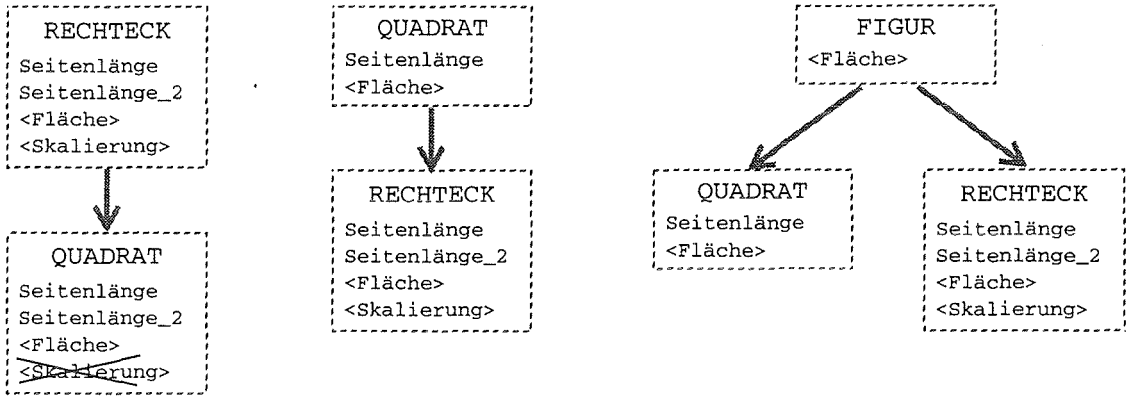


Abb. 10 Drei mögliche Vererbungsbeziehungen zwischen den Klassen RECHTECK und QUADRAT

Die Beziehung einer Unterklasse zu ihrer Oberklasse ist also nicht von vornherein festgelegt. Wenn auch versucht werden sollte, durch Vererbung eine Spezialisierung auszudrücken, so sind auch andere Bedeutungen möglich. Nach [Budd91] sind dies unter anderem:

- Spezialisierung: im Sinne von "ein Hund ist ein Säugetier", "ein PKW ist ein Kraftfahrzeug".
- Generalisierung: die Unterklasse besitzt eine größere Funktionalität wie die Oberklasse, siehe das Beispiel mit Rechtecken und Quadraten oben.
- Erweiterung: ähnlich Generalisierung, aber läßt die von der Oberklasse übernommenen Attribute und Methoden unverändert.
- Kombination: wenn die Eigenschaften mehrerer Oberklassen kombiniert werden sollen, kann von ihnen eine gemeinsame Unterklasse abgeleitet werden, welche die Attribute und Methoden der Oberklassen übernimmt. Man hat dann den Fall mehrfacher Vererbung vorliegen.

Eine genauere Untersuchung der Semantik von ist-ein Beziehungen, unabhängig vom Aspekt der Vererbung von Eigenschaften, findet sich in [Brachman83].

Hat-ein Beziehung zwischen Objekten

In ähnlicher Weise ist die Semantik der Beziehung von Objekten zu ihren Unterobjekten nicht a priori festgelegt. [Heuer92] unterscheidet folgende Arten von Unterobjekten:

- gemeinsam/privat: können Objekte nur in jeweils einem anderen Objekt als Unterobjekt vorkommen, so werden sie als private Unterobjekte bezeichnet. Unterobjekte, die in mehreren anderen Objekten vorkommen, sind gemeinsame Unterobjekte.
- abhängig/unabhängig: wenn ein Unterobjekt von der Existenz des ihn umgebenden Objektes abhängt, so ist es abhängig, andernfalls unabhängig.
- eingekapselt/nicht eingekapselt: wenn ein Unterobjekt nur auf der Ebene des ihn umgebenden Objekts sichtbar ist, so ist es eingekapselt. Kann man direkt auf es zugreifen, so ist es nicht eingekapselt.

Auch diese Differenzierungen geben die Bedeutung einer hat-ein Beziehung nicht vollständig wieder. In Abbildung 9 besteht eine hat-ein Beziehung zwischen einem Sensor und einem Verstärker, die eine Signalverbindung ausdrückt. Dies wird mit Hilfe eines gemeinsamen, un-

abhängigen, nicht eingekapselten Unterobjekts modelliert. Die Beziehung ist von der Semantik her asymmetrisch, der Sensor hat einen Verstärker, an den er Signale weitergibt, der Verstärker dagegen empfängt Signale vom Sensor.

Während hat-ein und ist-ein Beziehungen direkt objektorientiert modelliert werden können, ist dies für andere Arten von Beziehungen nicht der Fall. Ein möglicher Weg besteht darin, die Beziehungen selber wieder mit Klassen und Objekten zu modellieren. Dies hat auch den Vorteil, daß man die Methode der hierarchischen Abstraktion für die Modellierung der Beziehungen einsetzen kann. In [Specht91] wird ein Klassifizierungsschema für Beziehungen zwischen mechanischen Maschinenteilen angegeben, Abbildung 11 zeigt einen Ausschnitt daraus.

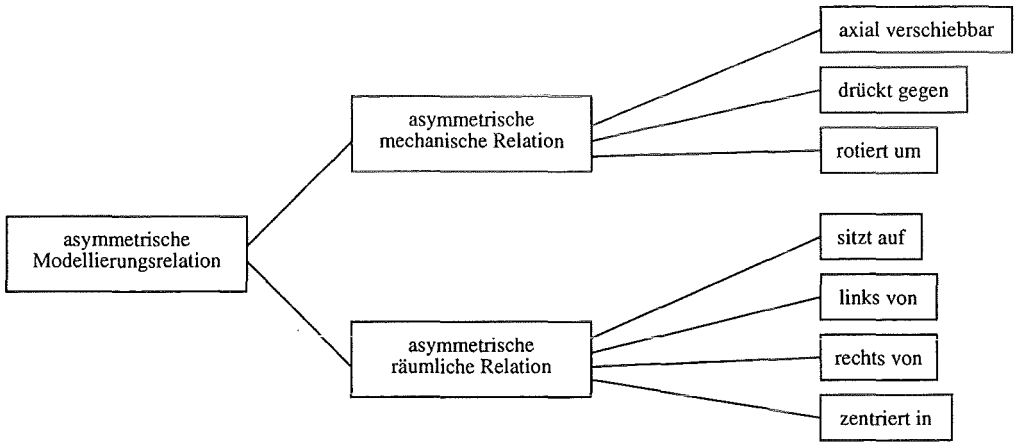


Abb. 11 Ausschnitt eines Klassifizierungsschemas für Beziehungen zwischen Komponenten einer mechanischen Anlage [Specht91].

Festlegen der Methoden

Nach der Festlegung von Klassen, Objekten und ihren Beziehungen untereinander können Methoden definiert werden, die das Verhalten der Objekte beschreiben. Hierbei wird es sich in vielen Fällen um vereinfachte Komponentenmodelle handeln. Man kann anhand der Implementierung auch testen, ob die Objekte richtig gewählt wurden: falls eine Methode zu oft auf Attribute anderer Objekte zugreifen muß, sollte möglicherweise die Objekteinteilung modifiziert werden.

3.1.4 Erweiterung der objektorientierten Methode

Auf Seite 26 wurde gezeigt, daß ein objektorientierter Ansatz die meisten Forderungen an die Modellierungsmethoden erfüllt, und daß nur die für ein Verständnis des Systems erforderlichen Erklärungen und heuristisches Wissen nicht adäquat wiedergegeben werden konnte. Es muß also zumindest noch eine andere Wissensrepräsentation für die Modellierung verwendet werden, die möglichst gut in eine objektorientierte Modellierung zu integrieren ist. Hierbei bieten sich, wie schon bemerkt, informelle Beschreibungen oder Regeln an.

Die Verwendung von Regeln zusammen mit Objekten bzw. Frames zur Wissensrepräsentation ist gängige Praxis in vielen aus der KI-Welt stammenden Werkzeugen (von denen eines im nächsten Unterkapitel vorgestellt wird). Die Integration beider Methoden ist dabei derart, daß auf Objekte innerhalb der Regeln zugegriffen wird, Regeln also die primäre Wissensrepräsen-

tationsform sind. Abgesehen davon, daß für diese Art der Integration kein Standard existiert, bleiben die prinzipiellen Nachteile von Regeln bei der Modellierung (siehe Seite 17) erhalten, daher scheidet diese Lösung aus.

Die andere Form der Integration beider Verfahren gliedert Regeln in die Objekte ein. Im Bereich der objektorientierten Analyse gibt es einen Ansatz, bei dem Regeln an Objekte gebunden werden [Graham93]. Eine Motivation besteht darin, die Absichten beim Erstellen der Analyse für Anwender klar, d.h. lesbar, auszudrücken. Auch hier existiert kein Standard, und generell gibt es bisher auch nur wenige konkrete Ansätze in der Forschung [Eick93] auf diesem Gebiet, so daß auch diese Kombination von Objekten und Regeln für Modellierungszwecke nicht geeignet ist.

Die Integration von informellen Beschreibungen in Objekte ist dagegen leicht möglich. Auf der Ebene von Klassen, Objekten und Attributen kann den Elementen der Modellierung eine informelle Beschreibung zugeordnet werden, in dem der Modellierer weiterführende Informationen und kritische Designentscheidungen beschreiben kann. Durch die Zuordnung erfolgt eine Strukturierung der gesamten informellen Information, außerdem wird die Möglichkeit des Zugriffs über die Objekte gegeben. Neben den bereits erwähnten Nachteilen informeller Beschreibungen (Seite 19) darf ein weiterer Nachteil nicht verschwiegen werden: die Konsistenz zwischen objektorientierter Modellierung und den Beschreibungen muß vom Anwender gewährleistet werden, nach jeder Änderung am Modell müssen auch die davon betroffenen Texte, Grafiken etc. daraufhin überprüft werden, ob die in ihnen enthaltene Information noch mit dem Rest des Modells übereinstimmt.

Fazit
Eine objektorientierte Modellierung, ergänzt um informelle Beschreibungen, erfüllt die in Kapitel 2 aufgestellten Forderungen an die Modellierungsmethoden. Es bleibt noch zu untersuchen, wie ein Modellierungswerkzeug auf der Basis dieser Methoden implementiert werden kann.

3.2 Implementierung eines Modellierungswerkzeugs

Wie schon in Kapitel 2 bemerkt, ist eine geeignete Methode zwar notwendig, aber alleine nicht hinreichend, um eine Modellierung durchzuführen. Für eine Anwendung ist es ebenso wichtig, wie die Methode umgesetzt wird und welche Hilfsmittel dem Benutzer dazu zur Verfügung stehen.

Die folgenden Untersuchungen gliedern sich im wesentlichen in zwei Teile. Zunächst wird diskutiert, welche kommerziellen Werkzeuge im Prinzip für die Durchführung einer objektorientierten Modellierung in Frage kommen. Danach wird das in dieser Arbeit verwendete Werkzeug dahingehend bewertet, inwieweit es die im letzten Kapitel aufgestellten Forderungen bereits erfüllt und welche Ergänzungen gegebenenfalls dazu noch notwendig sind.

3.2.1 Kategorien von Hilfsmitteln

Bei der Auswahl eines geeigneten kommerziellen Werkzeuges sind im Bereich der Objektorientierung die Teilgebiete Programmiersprachen, Datenbanksysteme und KI zu untersuchen. Da man nicht erwarten kann, daß alle Anforderungen erfüllt werden, wird man ein Werkzeug derjenigen Kategorie verwenden, welche die wichtigsten Anforderungen erfüllt.

Im Bereich der objektorientierten Programmierung gibt es eine Vielzahl von Sprachen mit teilweise recht unterschiedlichen Eigenschaften. Fast allen gemeinsam sind einige fundamentale Defizite, die Sprachen als einen ungeeigneten Ausgangspunkt für eine objektorientierte Modellierung erscheinen lassen:

- Fehlende Persistenz der Objekte: die in einer Sprache beschriebenen Objekte existieren nur für die Laufzeit des Programms. Da die Modellierung im wesentlichen aus Objekten besteht, müßte ein Mechanismus, welcher sie bei Beendigung des Programms speichert, selbst implementiert werden.
- Nicht vorhandene Schnittstelle zum Benutzer: Programmiersprachen bieten von sich aus keinerlei Hilfsmittel für die grafische Repräsentation von Klassen oder Objekten.[1]

Diese beiden Punkte reichen aus, OO-Sprachen als (alleiniges) Modellierungswerkzeug zu verwerfen.

Speziell die fehlende Objektpersistenz ist auch für andere Anwendungen sehr störend. Die Erweiterung von Programmiersprachen um persistente Objekte war eine logische Folge dieses Umstands und führte zur Entwicklung von objektorientierten Datenbanksystemen (OODBS).[2] Es existieren zur Zeit schon einige kommerzielle Implementierungen, die überwiegend auf der Sprache C++ aufbauen [Wieland95]. OODBS kombinieren die Vorteile einer OO-Sprache und die Möglichkeiten konventioneller Datenbanksysteme wie Speicherung einer großen Zahl von Objekten, Verwaltung von Zugriffsrechten, Speichermedien etc. Nachteile sind:

- Die Schnittstelle zum Anwender variiert sehr stark zwischen einzelnen OODBS.
- Speziell im Anfangsstadium einer Modellierung werden die definierten Klassen häufig modifiziert, teilweise entfernt und neue hinzugefügt. Klassen dienen bei OODBS als Grundlage des Datenbankschemas. Dieses kann bei vielen OODBS nur umständlich oder unter Verlust der alten Datenbestände geändert werden.
- Die Methoden der Objekte werden meist nicht innerhalb der eigentlichen Datenbank verwaltet.

Da die Entwicklung von kommerziellen OODBS erst vor wenigen Jahren begonnen hat, ist bei diesen Punkten noch mit einer Verbesserung zu rechnen. Für die im Rahmen dieser Arbeit durchzuführende Modellierung erscheint eine OODBS einerseits als überdimensioniert, was die Möglichkeiten der Speicherung großer Datenmengen angeht, als auch als zu unflexibel, um schnell verschiedene Modellierungsvarianten durchzuspielen.

Bessere Benutzerschnittstellen und mehr Flexibilität bei der Modellierung bieten Werkzeuge aus dem KI-Bereich, daher wird in dieser Arbeit ein solches Werkzeug verwendet. Zwischen einzelnen Implementierungen bestehen große Unterschiede, so daß eine pauschale Bewertung nicht sinnvoll erscheint. Für eine differenziertere Bewertung der für die Modellierung verwendeten Expertensystemshell Nexpert Object sei auf die folgende Darstellung verwiesen.

---

1. Hier stellt Smalltalk eine Ausnahme dar, da diese Sprache eng mit einer eigenen Entwicklungsumgebung verbunden ist, die auch einen Klassenbrowser zur Verfügung stellt.
2. Ein anderer Weg zu OODBS führt über die Erweiterung bestehender Datenbankkonzepte um objektorientierte Ergänzungen.

3.2.2 Die Expertensystemshell Nexpert Object

Nexpert Object ist eine hybride Expertensystemshell, die zur Wissensrepräsentation Regeln und Objekte verwendet [Nexpert91]. Eine umfassende Bewertung von Nexpert Object als Expertensystemshell findet sich in [Turner91]. Hier sollen die Aspekte diskutiert werden, die eine objektorientierte Modellierung unterstützen, also wird die Anwendung von Regeln nicht weiter betrachtet.

Vorteile von Nexpert Object
Die Objekte in Nexpert Object erlauben eine objektorientierte Modellierung von Sachverhalten. Für eine statische Beschreibung können Klassen mit Attributen definiert werden, von denen dann Objekte instanziiert werden. Die beschriebenen Beziehungen zwischen Klassen und Objekten (ist-ein, hat-ein, Instanz-von) werden unterstützt. Für eine dynamische Modellierung werden einfache Formeln (ein Beispiel findet sich im nächsten Kapitel), welche das Verhalten von Komponenten beschreiben, in Nexpert Object selbst berechnet, aufwendigerere Algorithmen können in der Sprache C implementiert und in Nexpert Object eingebunden werden.

Nexpert Object unterstützt direkt gemeinsame, unabhängige, nicht eingekapselte Unterobjekte. Andere Arten der Beziehungen zwischen Objekten und ihren Unterobjekten (siehe Seite 30) können auch modelliert werden, die Semantik muß dann aber durch den Anwender sichergestellt werden.

Das Erstellen von Klassen und Objekten wird durch eine grafische Oberfläche ermöglicht, die bereits bei der Eingabe Benutzerfehler abfängt. Ebenso steht ein Browser zur Verfügung (siehe Abbildung 12), mit welchem durch Klassen- und Objekthierarchien navigiert werden kann. Diese eingebaute Oberfläche muß je nach Anwendung bei Bedarf durch ein eigenes Interface ersetzt oder ergänzt werden (siehe dazu auch den nächsten Abschnitt).

Nexpert Object hat eine Programmierschnittstelle, über die auf alle Funktionen von anderen Programmen aus zugegriffen werden kann. Dies ermöglicht eine Kopplung mit einer anwendungsspezifischen grafischen Oberfläche und anderen Programmen.

Nexpert Object ist auf den gängigen Rechnertypen (PC, Macintosh, mehreren Workstationtypen, IBM-Mainframe) und Betriebssystemen (MS-DOS, UNIX, MVS, VMS) lauffähig. Im Rahmen der Arbeit war es auf einem Rechner SPARCsystem 300 unter dem Betriebssystem SunOS installiert.

Nicht verwendete Eigenschaften von Nexpert Object
Nexpert Object besitzt einige Erweiterungen des objektorientierten Ansatzes, wie er im ersten Teil dieses Kapitels vorgestellt wurde. Diese können zwar in bestimmten Situationen eine Hilfe bei der Modellierung sein, sie sind aber sehr spezifisch für Werkzeuge aus dem KI-Bereich oder Nexpert Object selbst. Diese Erweiterungen werden im folgenden identifiziert und bei der Modellierung nicht verwendet, um die durchgeführte Modellierung nicht von den Möglichkeiten eines speziellen Produktes abhängig zu machen.

Nexpert Object erlaubt es, die Vererbung von Attributen individuell zu regeln. Klassenabhängig können Attribute von der Vererbung an Unterklassen ausgeschlossen werden. Oberklassen können von Unterklassen erben. Weiter können Objekte Attribute an ihre Unterobjekte vererben. Die Semantik der Beziehung zwischen Ober- und Unterklasse sowie von Objekten

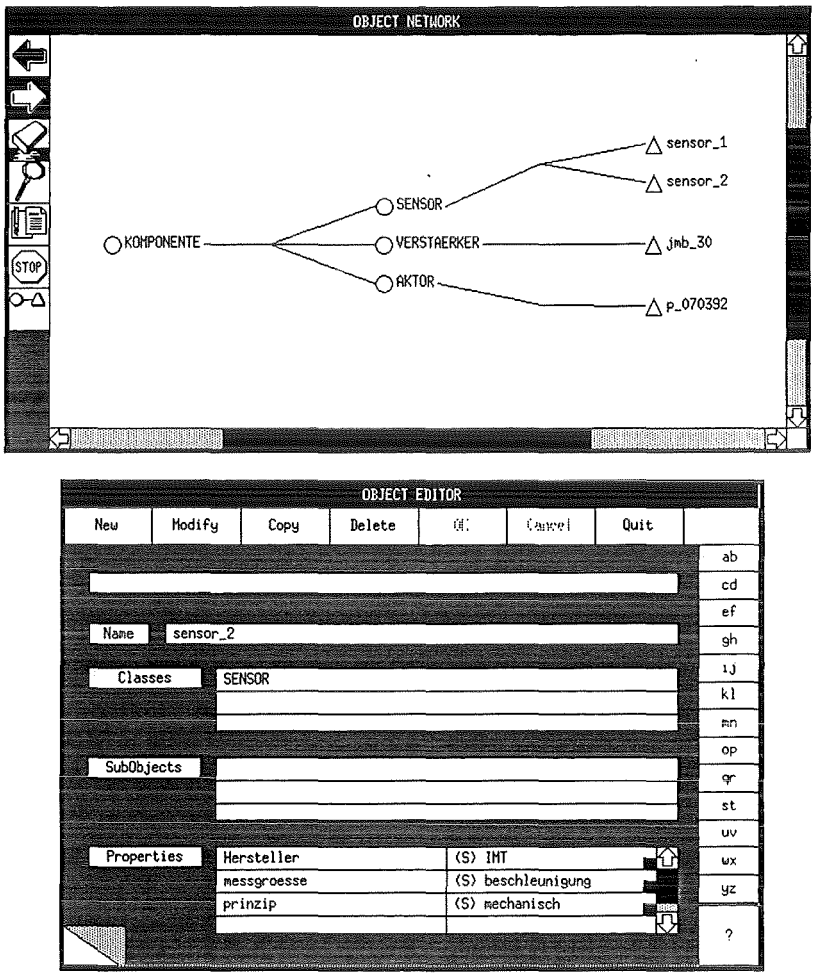


Abb. 12 oben: Klassen- und Objektbrowser von Nexpert Object, ○ zeigt Klassen, △ Objekte an.
unten: Objekteditor von Nexpert Object

zu ihren Unterobjekten wird dadurch aber völlig unbestimmt, daher wird zur Modellierung nur der schon beschriebene Vererbungsmechanismus zwischen Ober- und Unterklasse verwendet.

Es ist erlaubt, Objekte zu definieren, die keiner Klasse angehören oder die Instanzen von mehreren Klassen sind. Dies erspart in manchen Fällen die Definition von Klassen, bringt aber keine entscheidenden Vorteile. Daher wird davon kein Gebrauch gemacht.

Mängel von Nexpert Object

Wie jedes Werkzeug hat auch Nexpert Object Mängel, die nicht verschwiegen werden sollen, da hierin ein genereller, nicht zu vermeidender Nachteil beim Einsatz von kommerziellen Programmen deutlich wird: die Vorteile solcher Programme sind nicht ohne ihre spezifischen Nachteile zu erhalten. Bei Nexpert Object sind hier zu nennen:

- Idealerweise sollten alle Methoden, auch die in C geschriebenen, in Nexpert Object selbst verwaltet werden (allerdings geschieht dies in vielen anderen Werkzeugen, darunter auch den meisten OODBS, ebenfalls nicht).

- Es fehlt die Möglichkeit, Objekte in Datenstrukturen wie Feldern, Mengen oder Listen zu organisieren und über diese Strukturen dann auf sie zuzugreifen.
- Objektidentität wird über vom Anwender zu vergebende Namen hergestellt, d.h. zwei Objekte sind genau dann dasselbe Objekte, wenn sie den gleichen Namen haben. Dies erschwert es, komplexe Objekte mit Unterobjekten zu duplizieren, da für alle dabei neu entstehenden Objekte Namen vergeben werden müssen.

Und (erwartungsgemäß) hat Nexpert Object Defizite in anderen Bereichen, etwa bei der Darstellung von Informationen, da sich diese nur im Kontext einer konkreten Anwendung spezifizieren und lösen lassen.

Fazit
Nexpert Object unterstützt eine objektorientierte Modellierung ausreichend gut, um eine prototypische Implementierung eines Modellierungswerkzeugs darauf aufzubauen. Die festgestellten Mängel im methodischen Bereich können teilweise umgangen werden oder fallen nicht wesentlich ins Gewicht. Der Ausgleich der anwendungsspezifischen Defizite ist Gegenstand des nächsten Abschnitts.

3.2.3 Bewertung des Modellierungswerkzeugs

Für die Implementierung des Modellierungswerkzeugs ist zwischen aktiven und passiven Anwendern zu unterscheiden (womit bestimmte Rollen beschrieben sein sollen, keine Personen). Aktive Anwender modellieren selbst, definieren neue Klassen und neue Objekte, koppeln Komponentenmodelle an das Modell, haben also ein volles Zugriffsrecht auf alle Teile des Modells. Passive Anwender haben dagegen nur die Möglichkeit, das Modell zu inspizieren und einige temporäre Veränderungen vorzunehmen. Von aktiven Anwendern wird eine volle Beherrschung des Werkzeugs und der OO-Methode verlangt, während für passive Anwender dies nur zu einem geringen Umfang nötig ist. In der Praxis wird der Übergang zwischen aktivem und passivem Anwender fließend sein.

Nexpert Object unterstützt mit seiner Benutzerschnittstelle den aktiven Anwender bei der Definition von Klassen und Objekten, sowie dem Inspizieren des Modells auf einer sehr einfachen Ebene. Diese Schnittstelle muß um zusätzliche Funktionen für einen passiven Anwender erweitert werden. Im Rahmen dieser Arbeit werden daher Ergänzungen für eine Benutzerschnittstelle für einen passiven Anwender entwickelt, die wesentliche Teile der Forderungen aus dem letzten Kapitel (Seite 14 ff.) erfüllt. Als Richtschnur werden die in Kapitel 2 aufgestellten Forderungen herangezogen.

Erfüllung der Forderungen
Die angemessene Darstellung von Daten (Forderung 9) ist für alle denkbaren Arten von Daten in einer einzigen, allgemeingültigen Form nicht realisierbar. Exemplarisch wird als eigene Ergänzung für geometrische Daten eine veranschaulichende Darstellung mit einem Geometrieviewer (siehe Abbildung 28 auf Seite 60) implementiert. Die Position und Lage einzelner Komponenten des Mikrosystems kann so auf einen Blick erfaßt werden. Die Veranschaulichung von funktionalen Zusammenhängen zwischen Attributen geschieht durch die grafische Darstellung der Kennlinie (siehe Abbildung 29 auf Seite 61).

Nexpert Object erfüllt Forderung 10 nach Hilfe beim Zugriff auf das Modell bereits durch seine eigene Oberfläche (siehe Abbildung 12). Es werden vor allem aktive Anwender unterstützt, da hierbei ein ungehinderter Zugriff auf das Modell möglich ist. Für passive Anwender

wurde eine vereinfachte Version des Klassen- und Objektbrowsers implementiert, die ein unbeabsichtigtes Verändern des Modells ausschließt. Der Zugriff auf Klassen und Objekte erfolgt dabei im Gegensatz zu Nexpert Object nicht über den Anfangsbuchstaben des Namens, sondern über eine alphabetisch geordnete Liste (siehe Abbildung 13).

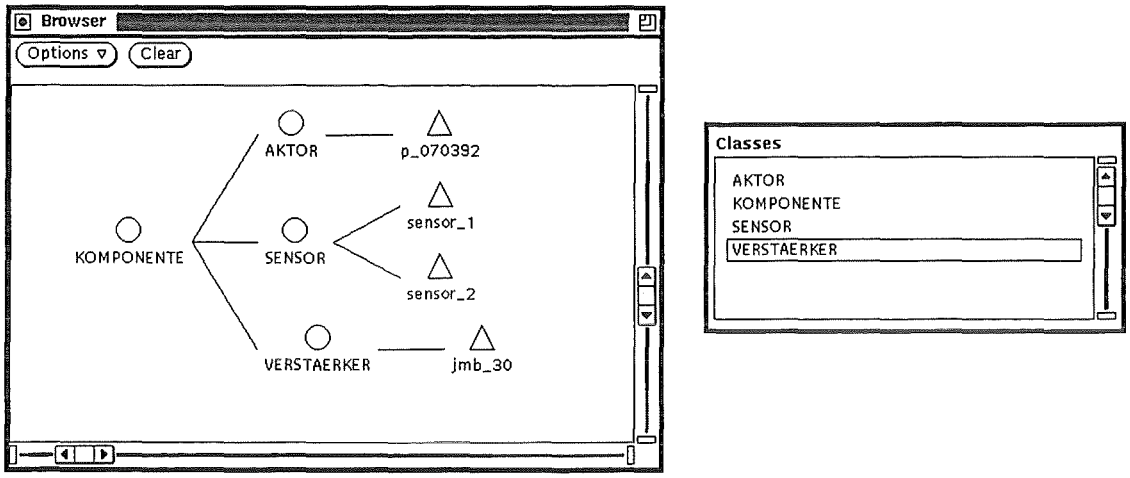


Abb. 13 Browser und Klassenliste der Benutzerschnittstelle für passive Anwender.

Für die Einbindung von Komponentenmodellen (Forderung 11) ist keine besondere Unterstützung notwendig, wenn sich die Komponentenmodelle als C-Routinen formulieren lassen und damit als Methoden integrierbar sind (siehe Kapitel 4). In ähnlicher Weise gilt dies für die Anbindung der Systemsoftware (Forderung 12). Auch sie läßt sich, wenn sie in C geschrieben ist, mit Nexpert Object integrieren. Einfache Algorithmen lassen sich sogar direkt als Methoden von Objekten einbinden. Weitere Aspekte der Einbindung von Komponentenmodellen werden in der Diskussion am Ende des Kapitels angesprochen.

Die Dezentralität des Modellierungswerkzeugs (Forderung 13) wird erreicht durch die Kombination von geeigneter Soft- und Hardware (siehe weiter unten), mit der die Benutzerschnittstelle implementiert wird. Dies ermöglicht es, auf die Modellierung von an ein Netzwerk angeschlossenen Rechnern so zuzugreifen, als ob sie lokal verfügbar wäre.

Forderung 14 nach Versionsverwaltung für das Modell wird von Nexpert Object nicht unterstützt. Zwar kann man mehrere Modelle gleichzeitig in einer Wissensbasis halten, aber der Anwender muß selbst wissen, welches Modell zu welchem Entwicklungsstand gehört. Die externen Routinen und auch die informellen Beschreibungen müssen ebenfalls separat verwaltet werden. Für die Versionsverwaltung kann entweder das Dateisystem des zur Modellierung verwendeten Rechners oder andere Hilfsmittel wie SCCS (Source Code Control System) verwendet werden. Wissensbasen von Nexpert Object, externe Routinen und Texte als Teil der informellen Beschreibungen lassen sich so gemeinsam verwalten.

Die Integration mit anderen am Mikrosystementwurf beteiligten Werkzeugen (Forderung 15) wird durch die Programmierschnittstelle von Nexpert Object ermöglicht. Weitere Überlegungen zur Werkzeugkopplung finden sich in der Diskussion am Ende des Kapitels.

Verwendete Hardware
Als Hardwareplattform dienen Server und Workstations der Firma SUN, die innerhalb eines Local Area Networks (LAN) auf Ethernet-Basis miteinander vernetzt sind. Als Betriebssystem wird SunOS, eine Variante von UNIX, verwendet. Sie bietet die Grundlage für einen dezentralen Zugriff auf das Modellierungswerkzeugs und ist erweiterungsfähig, d.h. es können auch PC's oder Rechner anderer Hersteller in die Umgebung eingebaut werden. Gleichzeitig sind damit auch die Anforderungen an die Grafikfähigkeiten der Hardware erfüllt, die sich aus der Notwendigkeit der geeigneten Darstellung von Daten ergeben. Es ist zu betonen, daß die beschriebene Hardwarekonfiguration den Stand der heute möglichen und auch bereits weit verbreiteten Technik repräsentiert. Die Anforderungen eines Modellierungswerkzeugs an die Hardware sind also ohne Schwierigkeiten zu erfüllen.

Verwendete Software
Für die Implementierung der Schnittstelle für passive Anwender wurde XView verwendet, eine Programmbibliothek zur Erstellung von grafischen Oberflächen unter dem X11 Window-Standard. Dabei wurden die Sprachen C++ und C eingesetzt. Daneben wurden weitere Programmpakete zur Verwaltung von Daten, zur grafischen Darstellung von Meßdaten und zur Entwicklung der Software für das Mikrosystem verwendet. Eine Aufstellung darüber findet sich im Anhang. Die meisten der eingesetzten Hilfsmittel sind für Forschungszwecke frei verfügbar und liegen als Quellcode vor. Da X11 der Standard für die Implementierung von Benutzeroberflächen auf Workstations aller namhaften Hersteller ist, wird es mit Sicherheit auch bei einer kommerziellen Implementierung eingesetzt werden. Dies gilt auch für die Sprachen C und C++.

Es ist wichtig zu betonen, daß die Implementierung des Modellierungswerkzeugs sich bei den verwendeten Hilfsmitteln bis auf Nexpert Object überwiegend auf Standards abstützt.

Eine konkrete Darstellung der Benutzerinteraktion mit dem Modellierungswerkzeug wird im nächsten Kapitel im Zusammenhang mit der Systemmodellierung gegeben. Dort werden auch die Teile der Benutzerschnittstelle erläutert, auf die bisher noch nicht eingegangen wurde.

3.3 Diskussion

In diesem Kapitel wurde die Methode der objektorientierten Modellierung vorgestellt und gezeigt, daß sie die in Kapitel 2 aufgestellten Forderungen erfüllt. Weiter wurde ein kommerzielles Werkzeug vorgestellt, welches diese Methode unterstützt. Erwartungsgemäß erfüllt dieses Programm die Forderungen an die Implementierung eines zur Mikrosystemmodellierung nur teilweise. Die verbleibenden Defizite werden teils durch eigene Ergänzungen behoben, teils wird ein Weg dazu aufgezeigt.

Im Folgenden wird zunächst eine mögliche Erweiterung der bisher dargestellten Modellierungsmethode vorgestellt. Danach wird die Anwendbarkeit der Methode auf andere Bereiche der Mikrosystemtechnik diskutiert. Schließlich soll das in diesem Kapitel konzipierte Werkzeug in den Kontext einer umfassenden Entwicklungsumgebung für Mikrosysteme gestellt werden, wobei der Gesichtspunkt der Kopplung mit anderen Werkzeugen eingehender beleuchtet wird.

3.3.1 Erweiterung der Modellierungsmethode

Die Darstellung des objektorientierten Ansatzes konnte, dem Rahmen dieser Arbeit angemessen, nur die wesentlichen Konzepte erläutern. Es ist durchaus denkbar, andere Methoden zu integrieren. Eine mögliche Erweiterung besteht z.B. darin, das Verhalten von Objekten mit Zustandsautomaten zu modellieren. Ein solches Vorgehen wird teilweise im Bereich des objektorientierten Design schon praktiziert [Booch91]. Dadurch wird es möglich, das dynamische Verhalten auf eine andere Art als durch Methoden zu beschreiben, die in einer konventionellen Programmiersprache abgefaßt sind. Allerdings wird nicht jeder Teil eines Systems durch Zustandsautomaten geeignet modelliert werden können (siehe dazu die Anmerkungen auf Seite 21). Eine solche Erweiterung ist nur dann sinnvoll, wenn die Modellierung mit Zustandsautomaten dynamisch ist, d. h. die Modelle müssen ausgeführt werden können. Es gibt bereits Ansätze in Bereich der Mikrosystemtechnik, wo hierarchische Zustandsautomaten für Modellierungszwecke und ein sie unterstützendes Werkzeug eingesetzt werden [Süß93b], allerdings nicht als Teil einer objektorientierten Modellierung.

3.3.2 Objektorientierte Modellierung in der Mikrosystemtechnik

Die Anwendbarkeit eines objektorientierten Ansatzes bei einer Modellierung beschränkt sich nicht auf die Beschreibung von gesamten Systemen. Auch die Geometrie von Mikrostrukturen kann durch Klassen- und Objekthierarchien beschrieben werden [Eggert94]. In das Geometriemodell werden die in einem mechanischen CAD-System konstruierten Strukturen importiert, wobei gleichzeitig Informationen über zu vermessende Teile der Struktur mit abgespeichert werden. Aus der Geometriewissensbasis heraus werden dann Eingabedaten für ein elektronisches CAD-System generiert, in welchem die gesamte Geometrie nur noch als eine Menge von Polygonzügen repräsentiert wird. Die eigentliche Semantik des Geometriewissens, also die Information über Relationen zwischen einzelnen Geometrieelementen, bleibt im objektorientierten Datenmodell erhalten.

Auch bei der Modellierung des LIGA-Fertigungsprozesses [Brauch94] stellte sich bei der Prüfung verschiedener Wissensrepräsentationsformen der objektorientierte Ansatz als der geeignetste heraus, wobei die Möglichkeiten zur Organisation komplexer Informationen als besonderer Vorteil genannt werden. Die für die konkrete Modellierung eingesetzten Werkzeuge entsprechen weitgehend den in dieser Arbeit verwendeten (Nexpert Object, C++, XView). Dabei wurde, um den Erfordernissen einer Prozeßbeschreibung besser gerecht werden zu können, als zusätzliches methodisches Hilfsmittel noch die Fuzzy Logik verwendet. Abbildung 14 zeigt eine vereinfachte Prozeßhierarchie, wobei für ein leichteres Verständnis die in der vorliegenden Arbeit verwendete Notation für Objekte angewendet wird.

Der Vorteil einer gemeinsamen Grundlage, nämlich der Objektorientierung, für die Modellierung ganz unterschiedlicher Abschnitte im Entwurfs- und Fertigungsprozeß für Mikrosysteme, wird deutlich, wenn man sich vor Augen führt, daß für eine durchgängige Rechnerunterstützung die einzelnen Entwurfswerkzeuge nicht isoliert stehen dürfen, sondern miteinander gekoppelt werden müssen. Diese Kopplung unterschiedlicher Werkzeuge im Rahmen von Entwurfsumgebungen ist nicht trivial, sie wird aber dadurch vereinfacht, wenn eine gemeinsame konzeptionelle Basis vorhanden ist. Dies wird im folgenden Abschnitt noch weiter motiviert.

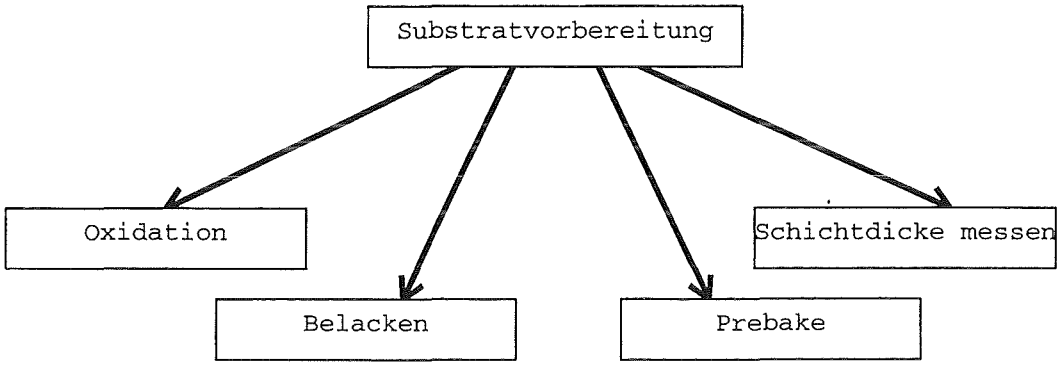


Abb. 14 Einfache Hierarchie von Objekten, die von einer gemeinsamen Klasse PROZEßSCHRITT abgeleitet sind (nicht angezeigt).

3.3.3 Integration von Werkzeugen in eine Entwicklungsumgebung

Die Erstellung und Anwendung von Werkzeugen für den Mikrosystementwurf ist Gegenstand des BMFT-Verbundvorhabens METEOR. Der multidisziplinäre Charakter der Mikrosystemtechnik bedingt eine hohe Zahl von Werkzeugen. Alleine bei den Partnern des Verbundprojektes sind über 90 verschiedene kommerzielle oder eigenentwickelte rechnergestützte Entwurfswerkzeuge im Einsatz [Müller-Glaser94]. Um ihre Integration nicht von vornherein als unlösbare Aufgabe erscheinen zu lassen, werden drei Ebenen von Entwurfsumgebungen definiert:

- "Entwurfsumgebung Gesamtsystem",
- interdisziplinäre "Entwurfsumgebung Systemintegration",
- "dedizierte Komponenten-Entwurfsumgebungen" (z.B. für elektronische Hardware, für Software, für mikromechanische und mikrooptische Komponenten etc.).

Aber auch auf einer Ebene ist die Zahl der Werkzeuge so groß, daß für ihre Integration u. a. die Entwicklung eines allgemeinen Objektschemas und eine geeignete Strukturierung der Daten auf der Basis eines CAE-Frameworks gefordert werden. Ein Modellierungswerkzeug, wie es in der vorliegenden Arbeit beschrieben wird, kann eng in eine "Entwurfsumgebung Gesamtsystem" integriert werden, da es eine strukturierte Systembeschreibung ermöglicht, die sich auf die objektorientierten Strukturen des Frameworks abbilden läßt. Dies gilt in ähnlicher Weise für die in [Brauch94] und [Eggert94] beschriebenen Ansätze.

Eine solche enge Kopplung ist allerdings nur dann möglich, wenn die Werkzeuge auf den gleichen Datenstrukturen arbeiten können. Dies ist in der Praxis oft nicht möglich, da insbesondere kommerzielle Werkzeuge ihre Daten in Eigenregie verwalten, ohne daß ein externer Zugriff möglich ist. In diesem Fall kann eine Kopplung nur über Konverterprogramme erfolgen, welche die Daten für einen In- und Export in eine geeignete Form überführen. Aufgrund der Vielzahl der bei der Entwicklung von Mikrosystemen eingesetzten Werkzeuge ist es nicht praktikabel, den notwendigen Datenaustausch der Werkzeuge jeweils untereinander durchzuführen. Es erscheint sinnvoller, eine zentrale Datenhaltung für alle bei der Entwicklung und Herstellung anfallenden Daten zu installieren, auf die dann die anderen Werkzeuge zugreifen.

Ein Datenaustausch zwischen Werkzeugen erfolgt dann immer über das zentrale Datenmodell, was die Zahl der zu implementierenden Schnittstellen minimiert. Abbildung 15 zeigt eine Kopplung zwischen verschiedenen CAD-Systemen und einem Vermessungssystem mit

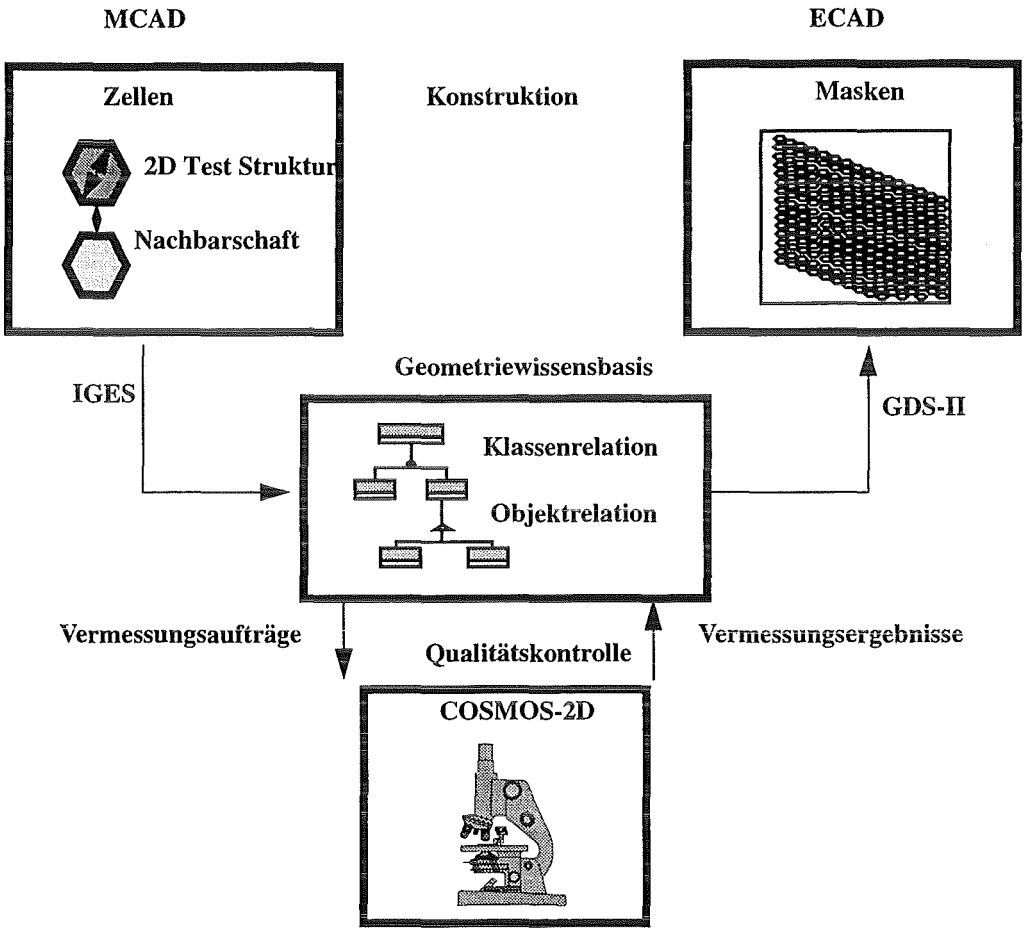


Abb. 15 Beispiel einer Werkzeugkopplung über eine objektorientierte Wissensbasis aus [Eggert94]. Die kommerziellen CAD-Systeme sind über Konverterprogramme an die Wissensbasis angekoppelt. Von dort hat ein Vermessungssystem Zugriff auf die Geometriedaten.

Hilfe einer objektorientierten Geometriewissensbasis. Prinzipiell ist für viele Bereiche eine solche Lösung realisierbar, wobei einige Nachteile in Kauf genommen werden müssen:

- die Programme, welche die Datenkonvertierung durchführen, müssen oft mit erheblichem Aufwand selbst entwickelt werden.
- Weiterentwicklungen des kommerziellen Herstellers sind durch Modifikationen am Konversionsprogramm nachzuvollziehen.
- für Anwender präsentieren sich die lose gekoppelten Werkzeuge als eigenständige Programme mit eigener Benutzerführung, deren Erlernung mit oft erheblichem Zeitaufwand verbunden ist.

Daher muß man bei der Konzipierung einer Entwurfsumgebung enge Maßstäbe anlegen, was die Zahl der eingesetzten Werkzeuge und die Art ihrer Kopplung angeht, da sonst der Erstellungsaufwand (zeitlich und finanziell) unverhältnismäßig hoch wird, während ihre Funktiona-

lität aufgrund der Komplexheit vom Anwender nicht ausgeschöpft werden kann. Eine Umgebung mit wenigen, eng gekoppelte Werkzeuge ist unter diesen Gesichtspunkten vielen, lose gekoppelten Werkzeugen vorzuziehen. Das hier entwickelte Konzept unterstützt dieses Vorgehen, da die Integration in ein objektorientiertes Framework direkt möglich ist.

Das Problem der Kopplung verschiedener Systeme und des Datenaustausches zwischen ihnen stellt sich natürlich nicht nur in der Mikrosystemtechnik, sondern überall dort, wo Produkte mit Rechnerunterstützung entwickelt und hergestellt werden sollen. Es gibt für Teilbereiche auch schon Lösungsansätze. Besonders interessant erscheint hier die Norm ISO 10303 : Product Data Representation and Exchange, die auch als STEP (STandard for the Exchange of Product model data) bekannt ist. Obwohl STEP noch nicht in der Lage ist, alle bei Mikrosystemen vorkommenden Arten von Daten (elektrische, mechanische, chemische, biologische, optische usw.) zu speichern und auszutauschen [Shaw93], sprechen langfristig eine Reihe von Vorzügen für seinen Einsatz auch in der Mikrosystemtechnik:

- STEP ist ein internationaler, herstellerunabhängiger Standard.
- STEP stellt mit EXPRESS eine strukturell objektorientierte Informationsmodellierungssprache zur Verfügung [Grabowski93], die zur Beschreibung von Produkten verwendet werden kann. Für eine Untermenge des Sprachumfangs steht mit EXPRESS-G eine grafische Notation zur Verfügung.
- Auf der Basis von EXPRESS definiert STEP ein Dateiformat, welches zum Speichern und Austauschen von Informationen verwendet werden kann.
- Durch die internationale Akzeptanz von STEP erscheint die Verfügbarkeit mit Werkzeugen, die diesen Standard unterstützen gesichert. Im CAD-Bereich sind Konverter von älteren Datenaustauschformaten wie z. B. IGES bereits kommerziell erhältlich.

Die Kopplung verschiedener Werkzeuge könnte dann so aussehen wie in Abbildung 16 ge-

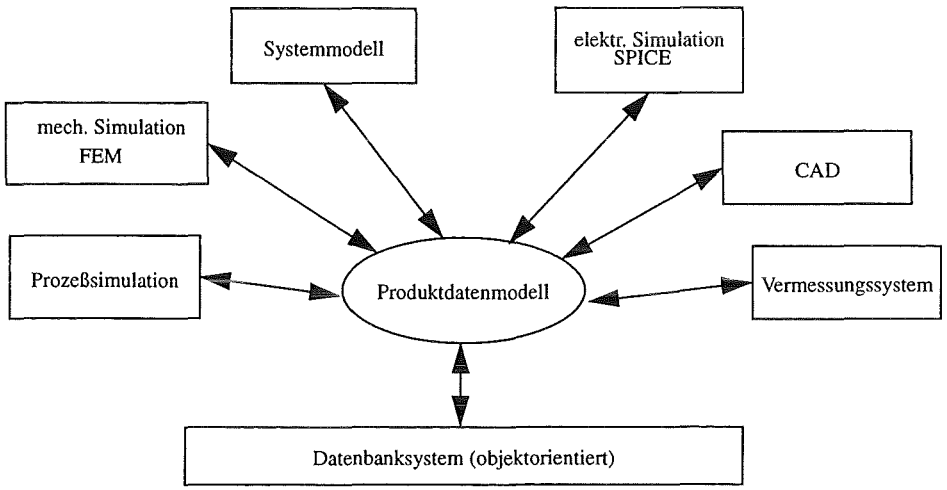


Abb. 16 Kopplung von Werkzeugen in der Mikrosystemtechnik durch ein Produktdatenmodell. Die verschiedenen Entwurfswerkzeuge greifen auf die im Datenmodell enthaltenen Infortionen zu, welches seinerseits auf einem Datenbanksystem aufsetzt.

zeigt. Die Realisierung ist allerdings erst langfristig möglich, weil die Normung eines vollständigen Produktdatenmodells in STEP noch nicht abgeschlossen ist.

4 Modellierung des Mikrosystems

In diesem Kapitel wird anhand eines existierenden Systems die Modellierung eines Mikrosystems mit Hilfe des entwickelten Werkzeugs durchgeführt. Dazu wird zunächst das reale System beschrieben, danach die Durchführung der Modellierung beschrieben und ein Vergleich zwischen realem System und Modell gezogen.

4.1 Das Mikrosystem zur Messung von Beschleunigungen

4.1.1 Beschreibung des Mikrosystems

Das reale Mikrosystem zur Messung von Beschleunigungen wird in einer Zusammenarbeit zwischen dem Institut für Mikrostrukturtechnik (IMT), der Hauptabteilung Prozeßdatenverarbeitung und Elektronik (HPE) und dem Institut für Angewandte Informatik (IAI) im Kernforschungszentrum Karlsruhe entwickelt [Strohrmann93a][Mohr94]. Abbildung 17 zeigt das Blockschaltbild des Mikrosystems. Innerhalb des Systems befinden sich sechs in LIGA-Tech-

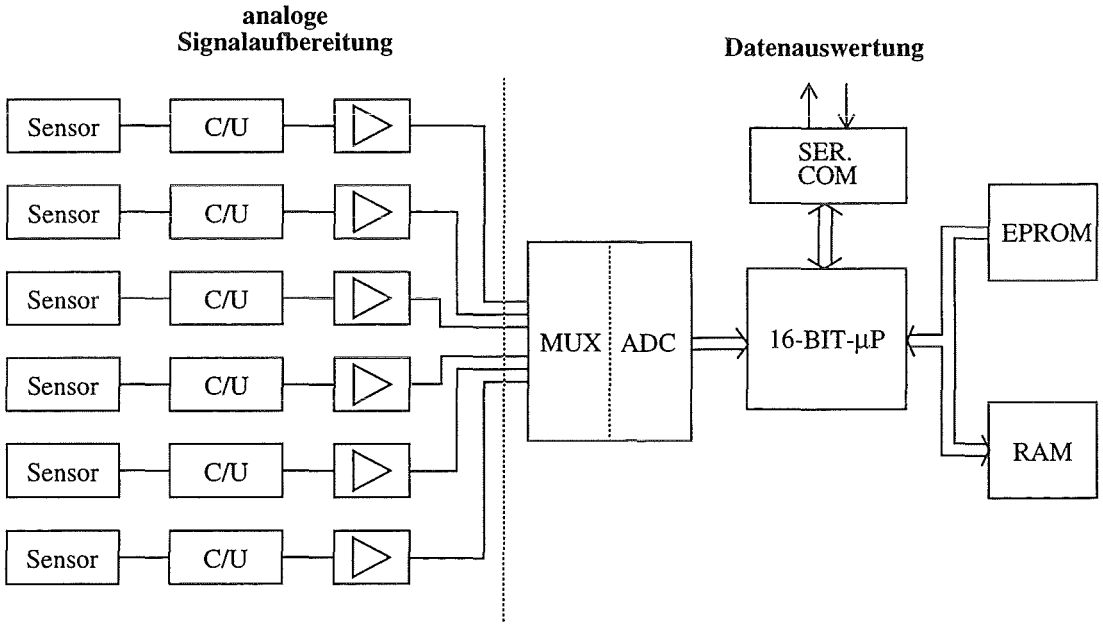


Abb. 17 Blockschaltbild des Mikrosystems zur Messung von Beschleunigungen [Strohrmann93a]. MUX (Multiplexer), ADC (Analog-Digital-Converter), SER.COM (serielle Schnittstelle) und µP (Mikroprozessor) sind physikalisch Teile eines Mikrocontrollers. C/U bezeichnet einen analogen ASIC, der die Kapazität des Sensors in eine elektrische Spannung umsetzt.

nik gefertigte Sensoren aus Nickel, die Beschleunigungen in zwei zueinander senkrechten Richtungen messen. Jeweils drei Sensoren messen also Beschleunigungen in einer Richtung. Der prinzipielle Aufbau eines Sensors ist einfach (siehe Abbildung 18). Am Ende einer Biegezunge schwingt eine seismische Masse zwischen zwei Elektroden und bildet mit diesen einen Doppelkondensator. Die Abbildung ist nicht maßstabsgetreu: für einen typischen Sensor beträgt die Länge der Biegezunge 185 µm, die Länge der seismischen Masse 3500 µm, die Spaltbreite eines Kondensators in Ruhelage dagegen nur 3 µm [Burbaum91]. Die Kapazität des Sensors wird zunächst durch einen integrierten Schaltkreis in eine Spannung gewandelt

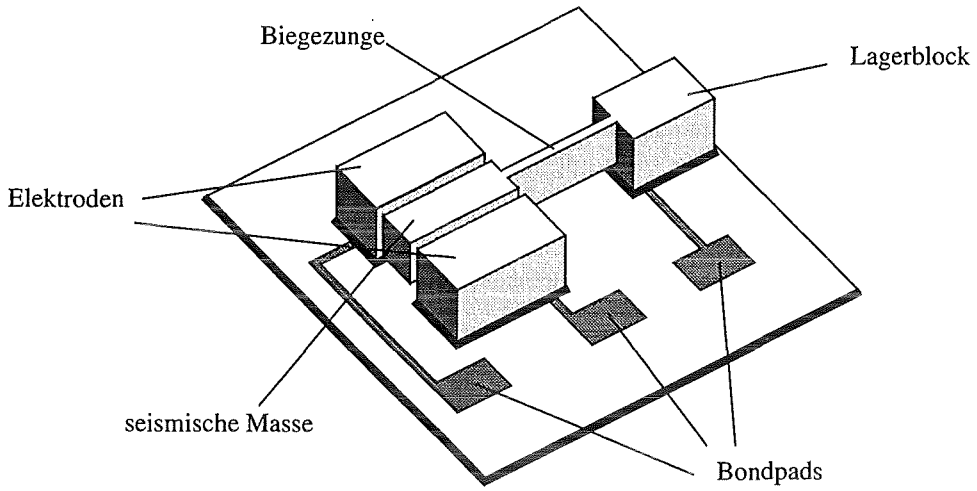


Abb. 18 Schematische Darstellung eines Beschleunigungssensors in LIGA-Technik.

und danach mit einem Instrumentenverstärker auf den Eingangsbereich des ADC angepaßt [Fromhein93]. Die Sensoren und die analoge Elektronik befinden sich in einem in Dickschichttechnik gefertigten Hybridmodul, welches auf ein in SMD-Technik realisiertes Mikrocontrollermodul gesteckt wird. Dieses enthält neben einem 16-Bit Mikrocontroller (SAB 80C166 [Siemens90]) noch Speicherbausteine (RAM, EPROM) sowie einige Bausteine zur Takterzeugung, zur Pegelanpassung für die RS232-Schnittstelle und zur Adreßdekodierung (nähere Beschreibung in [Fromhein93]). Der Mikrocontroller ist aufgrund seines Eingangsmultiplexers und eines internen 10-Bit ADC in der Lage, die als analoge Spannungen vorliegenden Sensorsignale zu digitalisieren.

Im Rahmen dieser Arbeit wurde für das Mikrosystem eine Software geschrieben, welche die digitalisierten Sensorsignale verarbeitet und über die serielle Schnittstelle mit einem anderen Rechner kommunizieren kann. Dabei besteht die Software aus zwei Teilen:

- Ein Kernel [Süß93a], welcher grundlegende Aufgaben der Systemsteuerung übernimmt (Datenaufzeichnung, Speicherverwaltung, Bereitstellen von Systemzeit etc.) und unabhängig von einer speziellen Anwendung ist. Für Testzwecke wurde der Kernel mit einer Kommunikationsschnittstelle versehen, über die seine Funktionalität kommandogesteuert getestet werden kann.
- Eine Demonstrationssoftware, welche stellvertretend für eine anwendungsspezifische Datenverarbeitung im Mikrosystem steht. Sie verwendet die vom Kernel bereitgestellten Dienste und muß daher nicht auf Hardwarekomponenten des Controllers direkt zugreifen. Die Demonstrationssoftware hat die Aufgabe, verschiedene Aufgaben der Datenverarbeitung in Mikrosystemen zu veranschaulichen. Sie umfaßt u.a. eine Korrektur von Nullpunktsverschiebung und Nichtlinearitäten der Sensoren und als Beispiel einer Anwendung eine Fast-Fourier-Transformation (FFT) der gemessenen Daten sowie eine programmierbare Schwellwertdetektion. Auch die Demonstrationssoftware besitzt eine Kommunikationsschnittstelle, über die sie mit einem Hostrechner Befehle und Daten austauschen kann.

Beide Teile der Software im Mikrosystem wurden in der Sprache C entwickelt. Die Demonstrationssoftware ist somit völlig unabhängig vom eingesetzten Controller und kann bei Bedarf leicht auf eine andere Rechnerarchitektur übertragen werden. Der Kernel ist weitgehend, aber nicht völlig unabhängig vom Controller, da er Hardwarekomponenten des Mikrocontrollers wie Timer, ADC oder serielle Schnittstelle direkt ansteuern muß.

Um mit dem Kernel oder der Demonstrationssoftware kommunizieren zu können, wurde jeweils ein Programm auf einem Hostrechner implementiert, welches eine grafische Benutzeroberfläche zur Verfügung stellt, die einem Benutzer das Senden von Befehlen an das Mikrosystem ermöglicht und gleichzeitig die vom Mikrosystem kommenden Daten grafisch darstellen kann. Gleichzeitig wurde ein Versuchsaufbau realisiert, bei dem das System mit Hilfe eines rechnergesteuerten Roboterarms durch Drehung im Schwerefeld der Erde definiert angeregt werden kann (siehe Abbildung 19). Damit ist es möglich, einzelne Funktionen

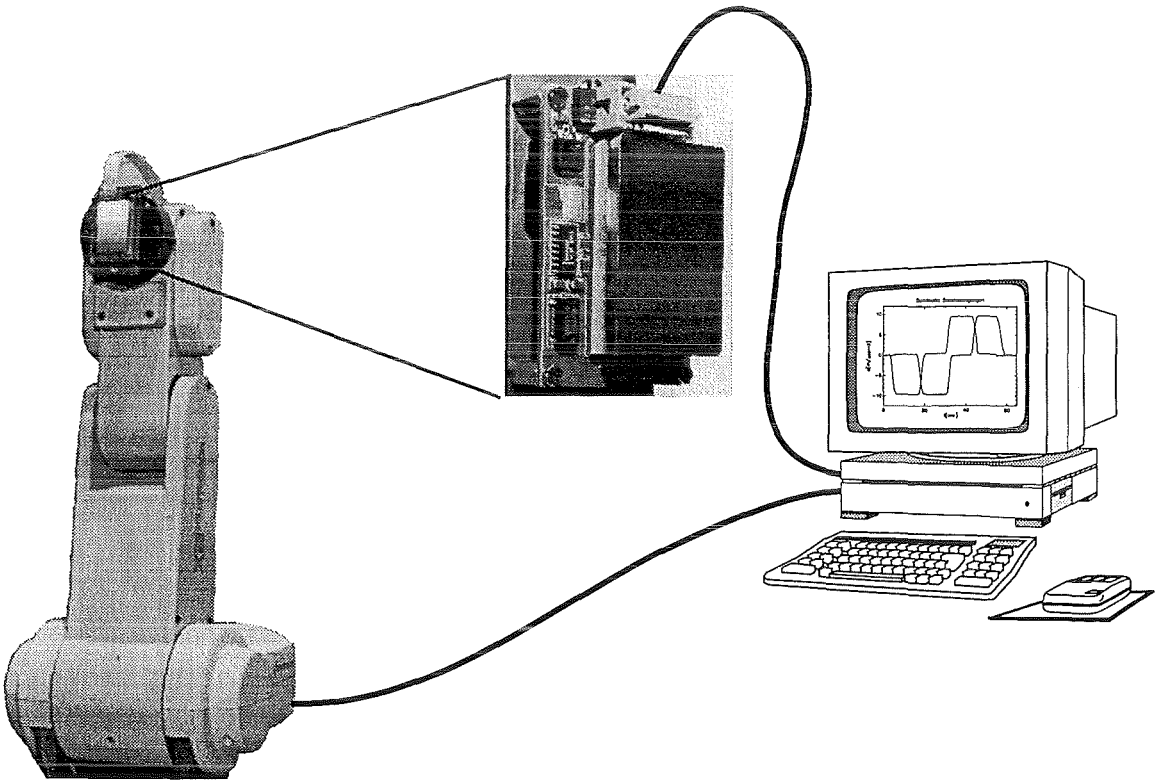


Abb. 19 Demonstrationsumgebung für Funktion des Mikrosystem [Lindemann94]. Gesteuert durch einen Rechner bewegt sich der Roboterarm. Die dadurch erzeugten Beschleunigungen werden vom System aufgezeichnet und zur Visualisierung an eine Workstation übertragen.

des Mikrosystems zu demonstrieren und auch Teile der Datenverarbeitung des Systems zu zeigen, die im Rahmen einer Anwendung verborgen bleiben, wie beispielsweise die Korrektur von Sensoroffsets und Nichtlineritäten von Kennlinien. Da hierbei auch das Funktionieren

einzelner Sensoren des Systems getestet werden kann (siehe Abbildung 20), ist hiermit die

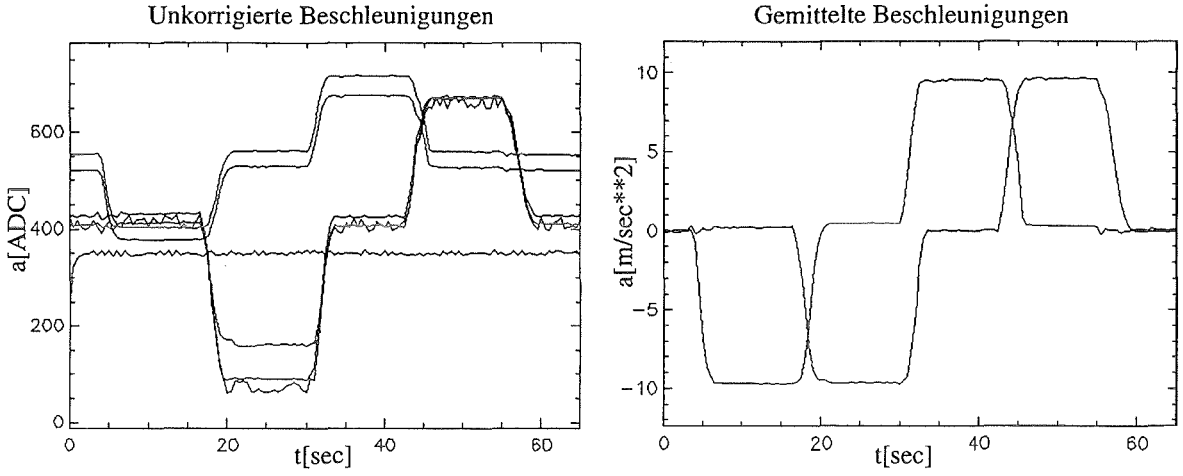


Abb. 20 Gemessene Beschleunigung bei Drehung des Mikrosystems im Schwerefeld der Erde. Links werden die unkorrigierten Daten der 6 Sensoren gezeigt, wie sie innerhalb des Systems in Form von digitalisierten Spannungen erfaßt werden. Deutlich ist zu erkennen, daß ein Sensor nicht angeschlossen ist und kein Signal liefert. Rechts werden die korrigierten und gemittelten Beschleunigungen gezeigt. Die systeminterne Datenverarbeitung blendet den nicht angeschlossenen Sensor aus der Weiterverarbeitung aus, so daß das System korrekte Meßergebnisse liefert.

Vorstufe zu einem Testsystem gegeben. Wie für das Modellierungswerkzeug wurde als Hardwareplattform eine Workstation der Firma Sun unter dem Betriebssystem UNIX eingesetzt. Diese Wahl ist aber nicht zwingend, da die Anforderungen an die Plattform, im Gegensatz zur Modellierung, hier wesentlich geringer sind. Als Hostrechner könnte also ebenso z.B. ein PC unter MS-DOS und Windows eingesetzt werden.

4.1.2 Bewertung des Mikrosystems

Da die Modellierung anhand eines konkreten Systems durchgeführt wird, erscheint es sinnvoll zu prüfen, welche mikrosystemtypischen Eigenschaften es aufweist. Bei der Untersuchung kommen die auf Seite 6 ff. aufgeführten Kriterien für Mikrosysteme zur Anwendung, wobei Zahl und Art der Komponenten und der Systemcharakter hier die entscheidenden Merkmale sind.

Die meisten wesentlichen Komponenten eines vollständigen Mikrosystems sind in dem betrachteten System vorhanden (man vergleiche dazu die Abbildungen 1 und 17). Zwar findet man nur einen einzigen Sensortyp, nämlich den Beschleunigungssensor, aber durch den vektoriellen Charakter der Beschleunigung und die geeignete Anordnung der Sensoren werden dennoch zwei voneinander unabhängige Meßgrößen durch das System erfaßt und ausgewertet. Mikrosystemtypisch ist auch das LIGA-Verfahren zur Herstellung der Beschleunigungssensoren. Durch "batch-processing" ist eine kostengünstige Fertigung von Sensoren möglich; dies ermöglicht den Einsatz von mehreren Sensoren (hier drei) für die gleiche Meßgröße, was eine Selbstüberwachung des Systems ermöglicht und durch die Redundanz eine erhöhte Ausfallsicherheit bewirkt. Als wesentliche Mikrosystemkomponenten fehlen lediglich die Aktoren. Es wurde aber schon bei der Diskussion von Komponenten von Mikrosystemen darauf

hingewiesen, daß sich die Entwicklung von Aktoren noch hauptsächlich im Forschungsstadium befindet. Insofern erscheint vertretbar, wenn sie im Rahmen einer Systemmodellierung noch nicht vorkommen.

Der Systemcharakter läßt sich an einer Reihe von Punkten nachweisen. Die Systemfunktion "Messung von Beschleunigungen" kann erfüllt werden, auch wenn einige Sensoren ausgefallen sind. Erreicht wird dies durch redundante Sensoren und durch die Datenverarbeitung, welche nur funktionsfähige Sensoren auswertet. Das Identifizieren defekter Sensoren kann auf verschiedene Weise erfolgen. Da mehrere Sensoren die gleiche Meßgröße erfassen, kann die Datenverarbeitung defekte Sensoren durch einen Vergleich von Sensoren untereinander erkennen. Falls dies nicht möglich ist (beispielsweise weil nur zwei Sensoren für die Messung eines Signals zur Verfügung stehen), kann die Funktion der Sensoren auch zu geeigneten Zeitpunkten durch einen externen Test überprüft werden. Die Fehlfunktion eines Sensors kann dem System dann per Kommando übermittelt werden. Die Korrektur von Offsets und Nichtlinearitäten der Sensoren geschieht sowohl durch eine geeignete analoge Auswertung der Kapazitäten (die Auswertung der Doppelkapazität nach Formel 6 (Seite 54) bewirkt eine lineare Abhängigkeit zwischen Auslenkung der seismischen Masse und der Ausgangsspannung) als auch durch die Datenverarbeitung; sie ist somit eine echte Systemfunktion und nicht nur an eine einzelne Komponente gebunden. Die Schnittstelle nach außen ist klar definiert; physikalisch durch die serielle RS-232 Verbindung, logisch durch eine Reihe von Kommandos, auf die das Mikrosystem mit entsprechenden Aktionen reagiert. Die Komplexität des Systems, verursacht u.a. durch die Sensoren, von denen jeder fertigungsbedingt seine eigene Charakteristik aufweist, ist nach erfolgter Sensorkorrektur nach außen nicht sichtbar.

Auf der Grundlage des vorliegenden Systems kann also eine Modellierung sinnvoll durchgeführt werden, da wesentliche mikrosystemspezifische Merkmale im vorliegenden System identifiziert werden können.

Einige Eigenschaften des Systems sind nicht charakteristisch für Mikrosysteme im allgemeinen. Zum einen kann die Rate, mit der Daten aufgezeichnet werden müssen, anwendungsbedingt sehr hoch sein (wenn schnelle Schwingungen aufgezeichnet werden müssen); sie kann über 1000 Messungen/sec betragen. Im Vergleich zu Systemen mit chemischen Sensoren, bei denen die Ansprechzeiten der Sensoren im Bereich von Sekunden oder Minuten liegen [Hoffmann93], ist das einige Zehnerpotenzen höher. Natürlich kann diese Datenrate im Modell nicht in Echtzeit simuliert werden. Allerdings stellt dies auch einen Extremfall dar, bei der Messung von quasi-statischen Beschleunigungen ist die Datenrate im realen System sehr viel niedriger, so daß auch das Modell hier vergleichbare Zeiten zur Simulation benötigt.

Zum anderen ist die Mikro-Makro-Schnittstelle oder auch IES-Kopplung (Information, Energie, Substanz [Menz93b]) des Systems zur Außenwelt relativ einfach. Der Austausch von Informationen geschieht über eine serielle Zweidraht-Verbindung, die Versorgung mit Energie erfolgt konventionell über einen elektrischen Leiter, ein Austausch von Substanzen findet nicht statt. Die Sensoren benötigen kein "Fenster" zum physikalischen Prozeß, da Beschleunigungskräfte unmittelbar übertragen werden. Durch eine geeignete Kapselung können sie daher gut gegen mögliche Störeinflüssen (Feuchtigkeit, chemische Verbindungen, elektrische Felder etc.) geschützt werden. Neben der zu messenden Größe Beschleunigung (oder Gravitation) ist dann die Temperatur die einzige Störgröße, die Auswirkungen auf das Systemverhal-

ten hat. Die IES-Kopplung des Systems mit seiner Außenwelt ist also vergleichsweise unkompliziert. Wie Aktoren befinden sich komplexere IES-Kopplungen vorwiegend im Stadium der Erforschung, so daß eine Modellierung noch nicht gut möglich ist.

4.2 Modellierung des Mikrosystems

Im folgenden wird die Modellierung des Mikrosystems durchgeführt. Dabei soll nicht nur diskutiert werden, welche Aspekte des realen Systems in die Modellierung einfließen, sondern auch, welche Alternativen bei der Durchführung der Modellierung gegeneinander abzuwägen sind.

4.2.1 Beschleunigungssensor

Es ist naheliegend, die sechs im System befindlichen Sensoren als Objekte für eine Modellierung zu identifizieren. Die genaue Festlegung der Klasse oder Klassen, von denen diese Objekte instanziiert werden, die Bestimmung der Attribute, die sie beschreiben und der Methoden, die ihr Verhalten beschreiben, bedarf aber einer genauen Abwägung.

Statische Modellierung
Dabei ist die Analyse eines älteren Modells (siehe Abbildung 21) für Beschleunigungssenso-

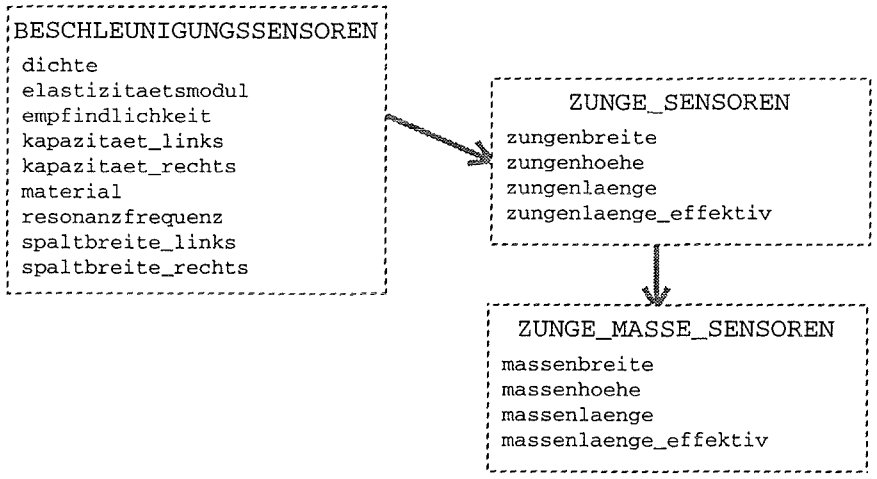


Abb. 21 Eine Klassenhierarchie für Beschleunigungssensoren aus [Lindemann91]. Dargestellt werden drei Klassen mit ihren jeweiligen Attributen.

ren hilfreich, die sich im wesentlichen auf die Beschreibung in [Burbaum91] stützt. Die abgebildete Klassenhierarchie modelliert ZUNGE_MASSE_SENSOREN (Sensoren mit seismischer Masse an einer beweglichen Biegezunge, siehe Abbildung 18) als Spezialisierung von ZUNGE_SENSOREN (Sensoren ohne seismische Masse, die Biegezunge bildet in diesem Fall mit den Elektroden die Kapazität). ZUNGE_SENSOREN wiederum ist von der Klasse BESCHLEUNIGUNGSENSOREN abgeleitet. Diese Klasse beschreibt Beschleunigungssensoren mit für sie typischen Eigenschaften wie Resonanzfrequenz oder Empfindlichkeit, ohne eine bestimmte Ausprägung festzulegen. Daher werden von BESCHLEUNIGUNGSSENSOREN auch keine Objekte gebildet, im Gegensatz zu den abgeleiteten Klassen, welche auch Attribute enthalten (z.B. zungenlaenge), die eine konkrete Implementierung beschreiben.

Im Kontext einer Systemmodellierung muß diese Klassenhierarchie überprüft werden. Wenn von vornherein ausgeschlossen werden kann, daß Sensoren der Klasse ZUNGE_SENSOREN im System eingesetzt werden (z.B. weil sie gewünschte Eigenschaften aus fertigungstechnischen Gründen nicht besitzen), ist diese Klasse nicht nötig und sehr wahrscheinlich sogar kontraproduktiv, weil sie die Modellierung komplexer macht, ohne einen Verständnisvorteil zu bringen. Der Name ZUNGE_MASSE_SENSOREN, der die Spezialisierung auch durch die Benennung deutlich machte, hat diese Funktion verloren und kann durch das kürzere LIGA_SENSOR ersetzt werden.

Ebenso sind die verwendeten Attribute darauf zu testen, ob sie nur die Komponente Sensor beschreiben, ob sie einen Einfluß auf das Systemverhalten als Ganzes haben oder ob ihr Vorhandensein zumindest zum Systemverständnis beiträgt. Falls ersteres der Fall ist, sollten diese Attribute gelöscht werden, da sie die Modellierung unnötig komplex machen. Im behandelten Beispiel verwendet LIGA_SENSOR zur Beschreibung der Länge der seismischen Masse zwei Attribute, massenlaenge (die geometrische Länge der seismischen Masse) und massenlaenge_effektiv (berücksichtigt, daß nicht die gesamte geometrische Länge zum Kondensatorspalt beiträgt). Diese Unterscheidung ist auf der Systemebene nicht mehr sinnvoll.

Betrachtet man die Attribute von LIGA_SENSOR, so bemerkt man, daß einige von ihnen inhaltliche Beziehungen haben. Dies sind z.B. dichte, elastizitaetsmodul und material, welche die Werkstoffeigenschaften beschreiben. Diese Zusammengehörigkeit kann man bei der Modellierung dadurch ausdrücken, daß eine Klasse MATERIAL_PARAMS eingeführt wird, die diese Attribute enthält. LIGA_SENSOR erhält diese Attribute nicht mehr durch Vererbung von ZUNGE_SENSOREN, sondern erbt sie von MATERIAL_PARAMS (unter Verwendung von mehrfacher Vererbung). Alternativ dazu ist es auch denkbar, die Werkstoffeigenschaften nicht als Attribute des Sensors, sondern im Rahmen eines Unterobjektes zu modellieren (siehe Abbildung 22), wobei dieses Unterobjekt dann nicht einem physikalischen Gegenstand entspricht, sondern nur der Strukturierung des Modells dient[1], um es verständlicher zu machen. Dieser Weg wurde in der Modellierung des Mikrosystems beschritten. Geometrische Eigenschaften wie die Fläche, die ein Bauteil einnimmt, werden auf eine analoge Weise modelliert.

Noch ein weiterer Aspekt läßt sich diskutieren: je genauer eine Modellierung wird und je komplexer die beschriebenen Komponenten sind, desto mehr Attribute werden zu ihrer Beschreibung benötigt. Im vorliegenden Beispiel hatte ZUNGE_MASSE_SENSOREN 17 Attribute, was bereits nicht mehr übersichtlich ist. In diesem Fall bietet es sich an, ein Objekt in weitere Unterobjekte aufzuspalten, und die Attribute in diesen anzuordnen. Im betrachteten Beispiel ist dies einfach: für den Sensor lassen sich sehr leicht die Komponenten seismische Masse und Biegezunge ausmachen (siehe Abbildung 18) und als Unterobjekte modellieren. Die Attribute zungenhoehe, zungenbreite etc. und massenhoehe, massenbreite etc. des Sensors fallen dann weg, dafür erhalten die neuen Unterobjekte jeweils Attribute hoehe, breite etc.

1. Objekte können also nicht nur gefunden, sondern bis zu einem gewissen Grad auch erfunden werden.

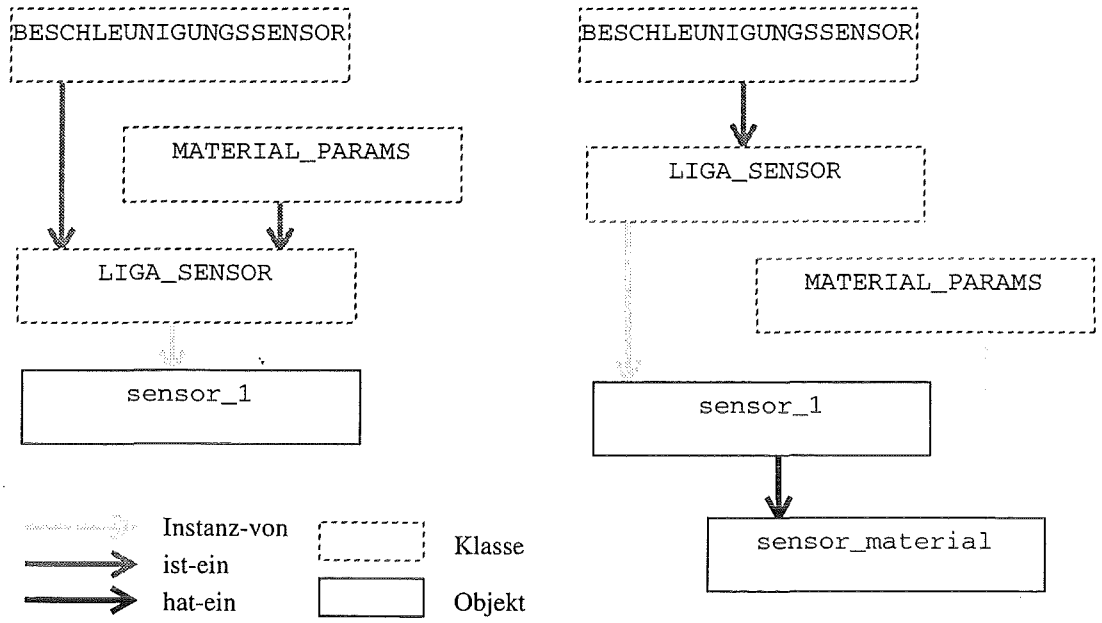


Abb. 22 Zwei Wege, die Materialparameter eines Sensors zu modellieren. Links: durch mehrfache Vererbung, rechts: durch Unterobjekt. Die Attribute der Klassen werden nicht aufgeführt.

Das Modell eines LIGA-Beschleunigungssensors auf Systemebene hat dann das in Abbildung 23 gezeigte Aussehen. Die im System vorhandenen sechs Beschleunigungssensoren unterscheiden sich (im Prinzip) nicht voneinander, die Modellierung der fünf nicht gezeigten Sensoren geschieht daher analog.

Modellierungsalternativen

Bei Betrachtung von Abbildung 23 fällt auf, daß die ist-ein Beziehung nur zwischen `BESCHLEUNIGUNGSSENSOR` und `LIGA_SENSOR` besteht. Es gibt also nur zwei Stufen der Abstraktion für die Sensoren. Teilweise ist dies dadurch zu erklären, daß ein konkretes System modelliert wird, bei dem die Sensorparameter zu Beginn der Modellierung schon feststanden. Dies wäre anders, falls Sensoren zu modellieren wären, bei denen die genaue Ausprägung noch nicht feststeht. Die Klassenhierarchie für die Sensoren des Systems könnte auch über drei Stufen gehen, wie in Abbildung 24 gezeigt. Die mittlere Abstraktionsstufe beschreibt dabei Sensoren, die zwar Beschleunigungen durch eine Änderung ihrer Doppelkapazität messen, bei denen aber das genaue Design noch nicht feststeht. Auch ein solcher Sensor kann leicht in das Modell einbezogen werden. Die Kapazitätsänderungen als Folge von simulierten Beschleunigungen müssen dabei durch den Anwender festgelegt werden. Er hat dabei mehr Freiheiten als bei dem konkreten `LIGA_SENSOR`, allerdings auch weniger Hilfen durch das Modell, unrealistische Parameterwerte zu erkennen.

Das Modell aus Abbildung 23 ist noch in zweifacher Hinsicht unvollständig:

- Bis jetzt fehlt noch dynamisches Verhalten, im Falle des Sensors ist dies eine Änderung der Kapazität bei anliegender Beschleunigung.
- Allgemeines Wissen oder zusätzliche erklärende Informationen über den Sensor sind ebenfalls noch nicht vorhanden.

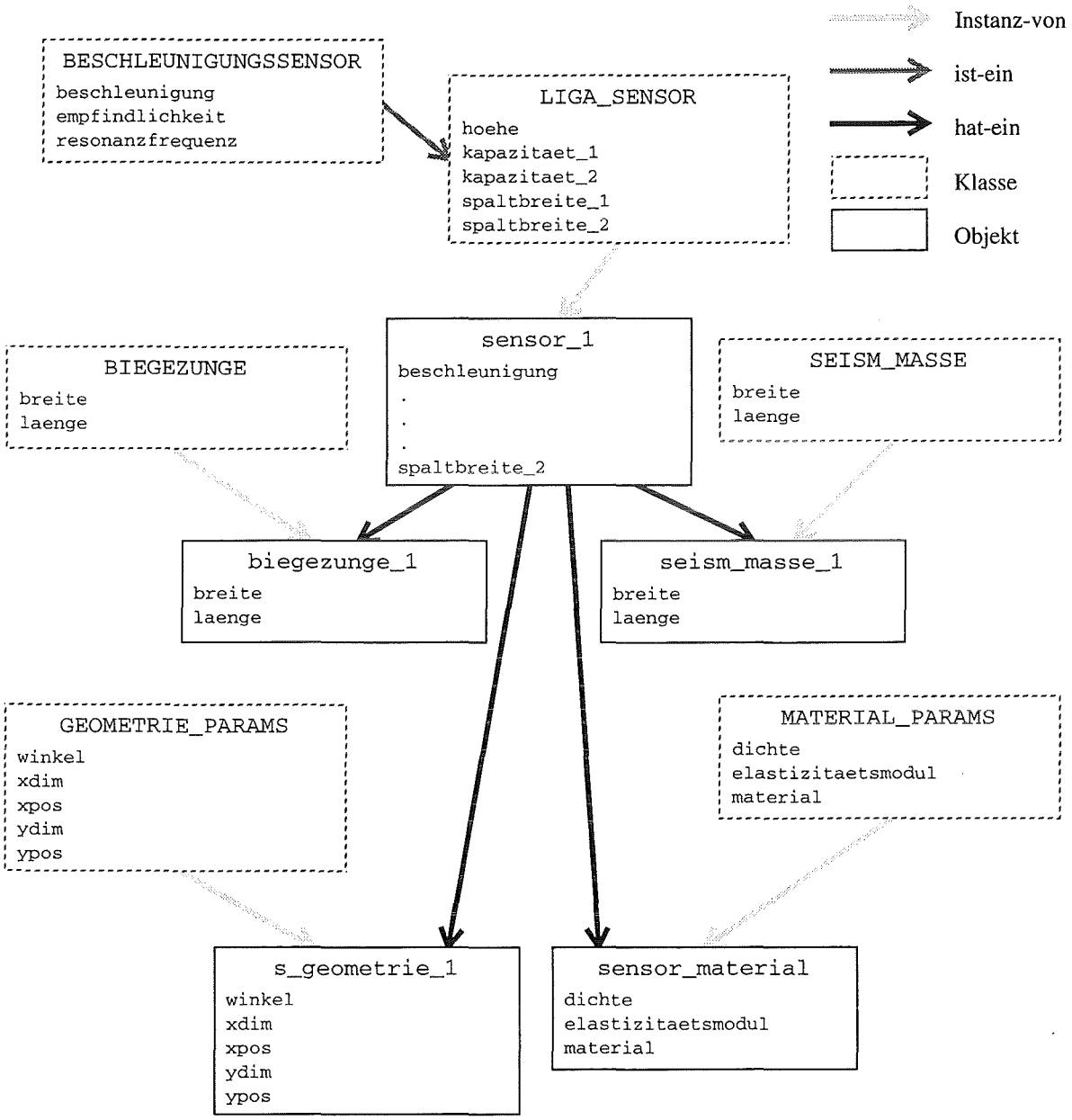


Abb. 23 Klassen- und Objekthierarchie des Modells des LIGA-Beschleunigungssensors. Bei der Auflistung der Attribute werden nur die jeweils neu in einer Klasse hinzukommenden angezeigt.

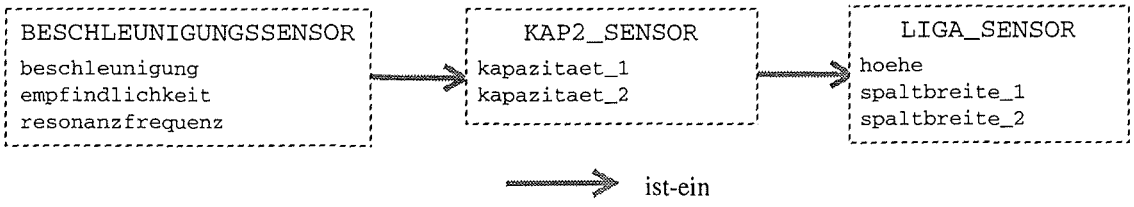


Abb. 24 Alternative Klassenhierarchie für Beschleunigungssensoren.

Dynamische Modellierung

Grundlage der dynamischen Modellierung ist die in [Burbaum91] vorgestellte Beschreibung des Sensorverhaltens, welchem im wesentlichen die Biegung elastischer Balken zugrunde liegt. Die Kapazitätsänderung errechnet sich dabei aus der Auslenkung der seismischen Masse, welche an einem Punkt x auf der seismischen Masse gegeben ist durch:

$$d_m(x) = d_z(l_z) + x \cdot \sin(\alpha(l_z)) \quad \text{wobei} \quad 0 \le x \le l_m \qquad \text{mit} \quad \begin{array}{ll} d_m: & \text{Auslenkung seism. Masse} \\ d_z: & \text{Auslenkung Ende Biegezunge} \\ l_z: & \text{Länge Biegezunge} \\ l_m: & \text{Länge seism. Masse} \\ \alpha: & \text{Winkel Ende Biegezunge} \\ x: & \text{Koordinate auf seism. Masse} \end{array} \tag{1}$$

Die Auslenkung am Ende der Biegezunge unter dem Einfluß einer Beschleunigung wird berechnet durch:

$$d_z(l_z) = \frac{a \cdot \rho \cdot l_z^4}{2 \cdot E_0 \cdot b_z^2}\left(3 + 8\frac{l_m b_m}{l_z b_z} + 6\frac{l_m^2 b_m}{l_z^2 b_z}\right) \qquad \text{mit} \quad \begin{array}{ll} a: & \text{Beschleunigung} \\ \rho: & \text{Dichte Material} \\ E_0: & \text{Elastizitätsmodul Material} \\ b_m: & \text{Breite seism. Masse} \\ b_z: & \text{Breite Biegezunge} \end{array} \tag{2}$$

und der Winkel der seismischen Massen am Ende der Biegezunge durch:

$$\alpha(l_z) = \frac{a \cdot \rho \cdot l_z^3}{2 \cdot E_0 \cdot b_z^2}\left(4 + 12\frac{l_m b_m}{l_z b_z} + 12\frac{l_m^2 b_m}{l_z^2 b_z}\right) \tag{3}$$

Die Kapazität des von der seismischen Masse und einer Gegenelektrode gebildeten Plattenkondensators läßt sich dann angenähert durch eine Summe berechnen, bei der die seismische Masse in N Teile geteilt wird:

$$C = \sum_{n=1}^{N} \varepsilon_0 \frac{l_m h}{N(s - d_m(x))} \quad \text{wobei} \quad x = n\frac{l_m}{N} \qquad \text{mit} \quad \begin{array}{ll} C: & \text{Kapazität des Kondensators} \\ \varepsilon_0: & \text{Influenzkonstante} \\ h: & \text{Höhe seism. Masse} \\ s: & \text{Kondensatorspalt ohne Beschleunigung} \end{array} \tag{4}$$

Diese Formeln lassen sich leicht als Programme in der Sprache C implementieren und als Methoden an das objektorientierte Modell in Nexpert Object anbinden. Wesentlich ist, daß die Kapazität des Kondensators schnell berechnet werden kann[1], und daß das Sensorverhalten trotzdem hinreichend genau wiedergegeben wird.

Auf die gleiche einfache Weise lassen sich auch Resonanzfrequenz und Empfindlichkeit bei gegebenen Sensorparametern berechnen:

---

1. Wobei schnell hier im Vergleich zu sehen ist mit rechenzeitintensiven Methoden wie FEM, mit denen eine genauere Abschätzung des Sensorverhaltens möglich ist, bei denen aber eine einzige Rechnung Zeit in der Größenordnung von Sekunden oder Minuten (abhängig von der Leistung des verwendeten Rechners und von der gewünschten Genauigkeit) benötigen kann.

$$E_g = \frac{C_g - C_0}{C_0}$$

$$\omega_0^2 = \frac{E_0 b_z^3}{12\rho b_m l_m l_z^3} \frac{2 + 6f + 6f^2}{2/3 + 4f + 11.5f^2 + 14f^3 + 8f^4}$$

mit

C_0: Kapazität bei $a = 0$
C_g: Kapazität bei $a = g$
E_g: Empfindlichkeit bei $a = g$
ω_0: Resonanzfrequenz
f: $\frac{l_m}{2l_z}$

(5)

Informelle Beschreibung

Obwohl jetzt der Sensor objektorientiert modelliert ist, ist die im Modell vorhandene Information nicht ausreichend, um ein volles Verständnis der Komponente Sensor zu ermöglichen. Diese zusätzliche Information, welche objektorientiert nicht oder nur schlecht vermittelt werden kann, soll einem Anwender in Form einer informellen Beschreibung zur Verfügung gestellt werden. Generell bieten sich mehrere Möglichkeiten an:

- Die Formeln, die das vereinfachte Komponentenmodell beschreiben und die im Modell nur als ausführbare Methoden enthalten sind, können explizit angegeben werden. Falls sie aus einem komplexen Komponentenmodell abgeleitet wurden, kann auch der Herleitungsweg skizziert werden. Dies kann dem Systementwickler helfen, die Grenzen des vereinfachten Modells besser abzuschätzen.
- Schematische oder photographische Darstellungen der Komponenten vermitteln schnell einen Überblick.
- Kritische Designparameter (im Falle des Beschleunigungssensors ist das vor allem die Spaltbreite des Kondensators) können explizit angesprochen und erläutert werden.

Beispiele dazu folgen in einem späteren Abschnitt dieses Kapitels. Die Entscheidung über Art und Menge dieser Informationen bleibt dem Modellierenden überlassen. Da sie bei einer Modifizierung der objektorientierten Modellierung ebenfalls immer geändert werden muß, wird sie zweckmäßigerweise umso umfangreicher ausfallen, je mehr sich eine Modellierung dem Abschluß nähert, weil dann die auftretenden Modifikationen mit hoher Wahrscheinlichkeit nicht mehr grundlegend sein werden, d. h. die Struktur der Klassenhierarchie ändert sich nicht mehr wesentlich.

4.2.2 Elektronik

Die wesentlichen elektronischen Bauteile sind der integrierte Schaltkreis zur Kapazitätsauswertung, der Instrumentenverstärker zur Pegelanpassung, der Mikrocontroller und die Speicherbausteine (RAM, EPROM). Diese Komponenten wurden nicht speziell für das Mikrosystem entwickelt, sondern entweder kommerziell erworben oder von Kooperationspartnern zur Verfügung gestellt.

Statische Modellierung

Wie im Fall der Sensoren ist es auch hier naheliegend, die Bausteine als Objekte zu modellieren. Im Falle des Mikrocontrollers ist ein Objekt aber nicht ausreichend, da er mehrere sehr unterschiedliche Funktionen wahrnimmt (Wandlung analoger Spannungen in digitale Werte, Abarbeiten der Software). Es erscheint daher richtiger, diese als getrennte Objekte zu modellieren, wie auch in Abbildung 17 die Funktion MUX/ADC als eigenständige Einheit gezeigt wird. Andererseits können Eigenschaften wie Flächenbedarf oder elektrische Leistungsauf-

nahme nur dem Mikrocontroller als Ganzes zugeordnet werden. Die Lösung besteht darin, den Controller als Objekt mit Unterobjekten zu modellieren, wobei die wesentlichen Eigenschaften des Controllers in seinen Unterobjekten enthalten sind.

Die Tatsache, daß die elektronischen Bauteile fertig entwickelt im System eingesetzt werden, hat auch Auswirkungen auf die Modellierung. Während es bei den Sensoren noch sinnvoll war, sie durch viele Attribute zu beschreiben, deren Werte man relativ frei variieren kann, ist dies bei nicht veränderbaren Bausteinen nur noch bedingt sinnvoll, da eine Veränderung eines Parameters in der Realität sehr oft mit dem Auswechseln des gesamten Bausteins verbunden ist. Eine große Anzahl von Attributen erfüllt dann überwiegend den Zweck der Dokumentation, dafür ist aber die informelle Beschreibung oftmals eine bessere Ausdrucksform (siehe unten). Daher sind für die Modellierung der Elektronik weniger Attribute als für die Modellierung der mikromechanischen Sensoren erforderlich, obwohl die elektronischen Bausteine selbst komplexere Komponenten sind[1].

Abbildung 25 zeigt die Klassen und Objekte des Modells des analogen Teils der Elektronik. Der digitale Teil besteht im wesentlichen aus dem Mikrocontroller und den Speicherbausteinen. Noch mehr als im analogen Teil hat hier das Modell im wesentlichen einen dokumentierenden Charakter, da hier gar keine Parameter im realen System mehr verändert werden können. Daher wird sie hier nicht explizit aufgeführt.

Dynamische Modellierung

Die Funktionalität von Bauteilen wie Operationsverstärker oder ADC kann durch einfache Funktionen beschrieben werden, die ein Eingangsspannungsintervall auf ein Ausgangsspannungsintervall oder (im Fall des ADC) auf ein Intervall von natürlichen Zahlen abbilden. Diese Methoden sind so einfach, daß sie mit den internen Hilfsmitteln von Nexpert Object formuliert werden können, d.h. es sind keine externen, in C geschriebenen Routinen dazu notwendig.

Die Auswertung der Doppelkapazität des Beschleunigungssensors durch den integrierten Schaltkreis geschieht nach folgender Formel [Leuthold90]:

$$U(x) = \frac{C_1(x) - C_2(x)}{C_1(x) + C_2(x)} U_0 \quad \text{mit} \quad \begin{array}{rl} U: & \text{Spannung am Ausgang} \\ U_0: & \text{Referenzspannung} \\ C_1, C_2: & \text{Kapazitäten des Doppelkondensators} \\ x: & \text{Auslenkung der seismischen Masse} \end{array} \tag{6}$$

Auch sie ist so einfach, daß sie innerhalb von Nexpert Object berechnet werden kann.

Wie bei den statischen Eigenschaften zeigt sich auch hier, daß das Verhalten von komplexen Komponenten wie dem integrierte Schaltkreis auf Systemebene nicht notwendigerweise mit einem erhöhten Aufwand bei der Modellierung einhergeht. Oft ist es vielmehr so, daß bei Komponenten ein erhöhter Aufwand mit zusätzlicher Komplexität im Innern zu dem Zweck betrieben wird, daß die Komponente von außen durch eine einfache Modellvorstellung richtig beschrieben wird[2].

---

1. Speziell der Mikrocontroller ist als Komponente komplexer als das Mikrosystem selbst und besteht aus einer Vielzahl von Einzelkomponenten, die überwiegend nicht modelliert werden.
2. Wie dies beim Mikrosystem als Ganzes auch geschieht, die Komplexität im Innern ist von außen nicht sichtbar (Seite 7).

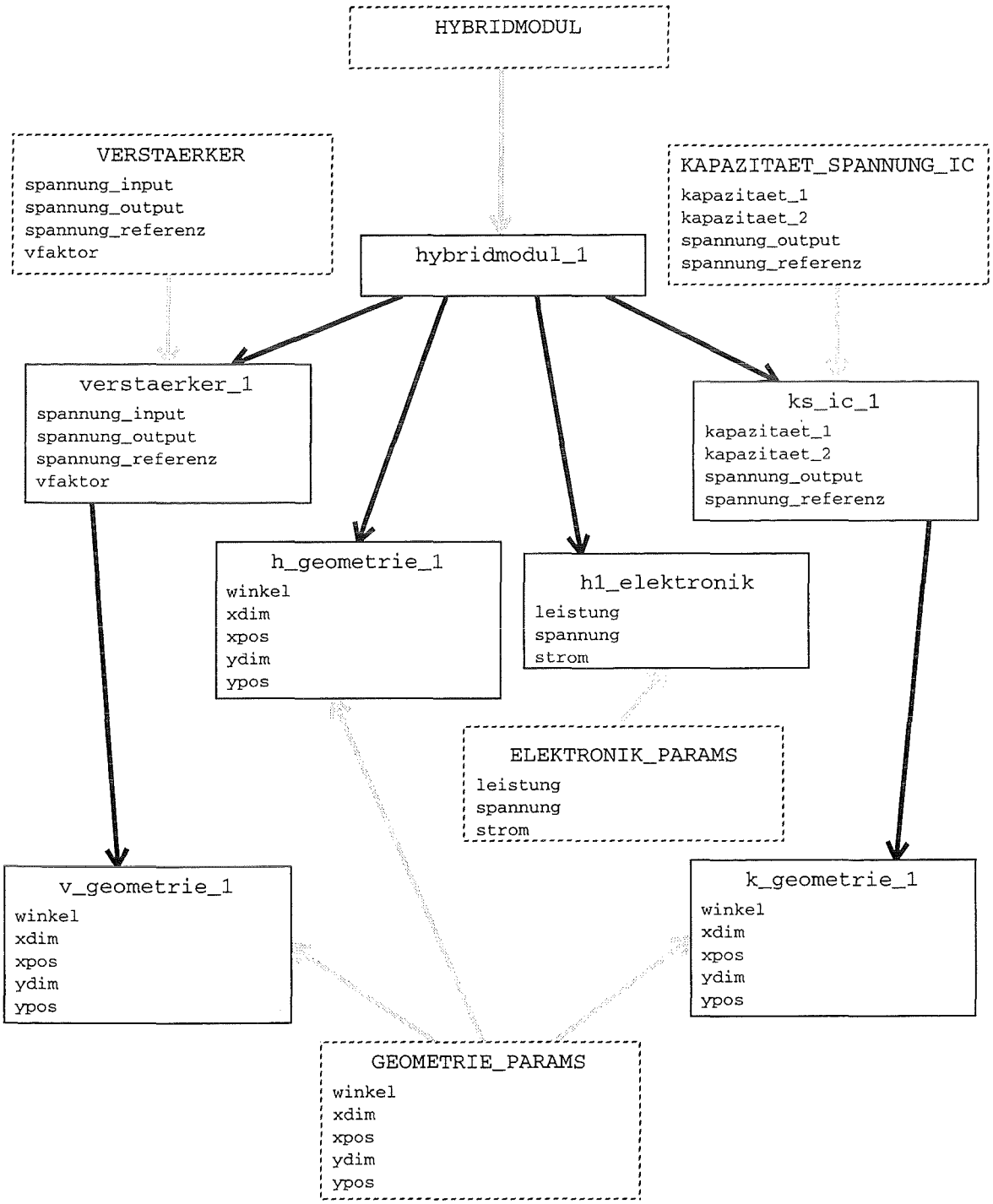


Abb. 25 Klassen- und Objekthierarchie des Modells des analogen Teils der Elektronik, wobei nur jeweils ein Objekt von jeder Klasse gezeigt wird. Sensoren als Unterobjekte von hybridmodul_1 werden ebenfalls nicht gezeigt.

Man hat beim Einsatz kommerzieller Elektronik nicht den Fall vorliegen, daß bei der Modellierung Komponentenmodelle eingesetzt werden, die das Verhalten eines idealen Bausteins annähern. Vielmehr sind diese Komponenten oft daraufhin optimiert, eine einfache Modellvorstellung (z.B. idealer Verstärker, ideale Kapazitäts-Spannungswandlung) möglichst genau in der Realität wiederzugeben. Dies widerspricht zwar dem Systemgedanken, daß die Teil-

komponenten wegen des erhöhten Aufwands nicht einzeln optimiert werden sollen (siehe dazu die Anmerkungen zum Systemcharakter von Mikrosystemen auf Seite 7). Da dieser Aufwand aber bei kommerziellen Komponenten schon vor Beginn der Systementwicklung geleistet, kann er in Kauf genommen werden[1].

Dem Prozessor, den Speicherbausteinen und weitgehend auch der seriellen Schnittstelle kann man im Rahmen der dynamischen Modellierung nicht ohne weiteres Methoden zuordnen, die ihr Verhalten bestimmen, da die auf dem Prozessor ablaufende Software die eigentliche Komponente ist, welche Funktionalität und Verhalten bestimmt. Diese Thematik wird unten bei der Modellierung von Datenverarbeitung und Kommunikation aufgegriffen.

Informelle Beschreibung
Wie oben bei der statischen Modellierung schon erwähnt, kommt der informellen Beschreibung bei fertigen Bauteilen verstärkt die Aufgabe zu, die jeweilige Komponente zu dokumentieren. Bei kommerziellen Bausteinen kann das im Extremfall darauf hinauslaufen, das jeweilige Datenblatt oder zumindest Teile davon in die Beschreibung aufzunehmen. Dies wird aber die Ausnahme bleiben, da ein Datenblatt oft viele Informationen enthält, die auf der Systemebene keine Rolle spielen. Weiter kann dokumentiert werden, welche Überlegungen zur Auswahl der Komponente geführt haben und welche Nachteile in Kauf genommen werden mußten. Ebenso kann erläutert werden, welche Alternativen aus welchen Gründen verworfen wurden oder ob gleichwertige andere Lösungen existieren.

4.2.3 Datenverarbeitung und Kommunikation

Die auf dem Prozessor ablaufende Software implementiert die Datenverarbeitung im System und die Kommunikation mit seiner Umwelt. Sie bestimmt primär das Verhalten des Systems als Ganzes, gleichzeitig ist sie die Komponente, welche am leichtesten während der Entwicklung des Systems geändert werden kann. Schon durch ihren immateriellen Charakter unterscheidet sie sich deutlich von anderen Komponenten.

Statische Modellierung
Trotzdem ist es möglich, die Software als Objekt in das Modell aufzunehmen. Wegen ihres immateriellen Charakters gibt es allerdings nur wenige statische Eigenschaften, die mit Attributen modelliert werden können. Denkbar sind benötigter Speicher (RAM, EPROM) oder die maximale Geschwindigkeit, mit der Daten verarbeitet werden können. Diese Angaben dienen primär der Dokumentation, wobei der Hauptteil der Dokumentation in informeller Form vorliegen wird, und die Modellierung als Objekt hier vor allem den Zweck hat, den Zugriff auf diese Dokumentation zu ermöglichen.

Dynamische Modellierung
Bis jetzt wurde die dynamische Modellierung mit Hilfe von Komponentenmodellen durchgeführt, die als Methoden von Objekten implementiert wurden. Im Fall der Software ist diese Vorgehensweise aus folgenden Gründen zu überdenken:

- Es ist nicht immer eindeutig, wie ein vereinfachtes Modell der Software aussehen kann. Soll es wichtige Algorithmen der richtigen Software enthalten, sollen die richtigen Routinen durch weitgehend funktionslose Dummyversionen ersetzt

1. Allerdings können auch selbstentwickelte Komponenten optimiert werden, wenn dies fertigungstechnisch keinen zusätzlichen Aufwand bedeutet (siehe dazu im nächsten Kapitel alternatives Sensordesign).

werden? Dies kann nicht allgemein beantwortet werden, es hängt u.a. davon ab, wo der Schwerpunkt der Datenverarbeitung liegt. Werden Daten hauptsächlich verwaltet, oder sind aufwendige Rechenverfahren zur Signalverarbeitung implementiert?

- Der Nutzen eines vereinfachten Modells ist ungewiß. Bei den Sensoren und Teilen der Elektronik diente das vereinfachte Modell dazu, eine prinzipielle Funktion verständlich zu machen und rechenzeitintensive Berechnungen zu vermeiden. Beides trifft bei der Software so nicht zu.

Dagegen kann ein möglicher Weg zur dynamischen Modellierung auch darin bestehen, die tatsächlich im Mikrosystem eingesetzte Software mit der Modellierung des Systems zu koppeln, und dadurch das Verhalten des Systems weitgehend zu simulieren (Forderung 12 auf Seite 15).

Um diese Kopplung durchführen zu können ist es notwendig, die Software des Mikrosystems zusammen mit dem Modell ablaufen zu lassen. Der Controller des Mikrosystems wird aber in der Regel nicht mit dem Prozessor des Rechners, auf dem das Modell abläuft, identisch sein. Folgende Wege zur Lösung dieses Problems sind möglich:

- Die Software kann auf einem per Software simulierten Controller ausgeführt werden (Emulation). Vorteile dieses Verfahrens sind:
  - Die Software des Mikrosystems kann unverändert eingesetzt werden, unabhängig davon, in welcher Form sie vorliegt oder in welcher Sprache sie geschrieben wurde.
  - Das durch die Software bestimmte Verhalten des Mikrosystems kann optimal simuliert werden, da auch Hardwarekomponenten des Controllers (z.B. Timer) simuliert werden.

  Als Nachteile dieses Ansatzes sind zu nennen:
  - Er ist nur durchführbar, wenn tatsächlich eine Simulation des Controllers vorhanden ist (es wird praktisch unmöglich sein, eine solche Simulation selbst zu erstellen, sie muß kommerziell verfügbar sein).
  - Der Typ des Controllers im Mikrosystem wird festgelegt.
- Die Software des Mikrosystems wird auf den Rechner portiert, auf dem das Modell abläuft, und dort ausgeführt. Das hat folgende Vorteile:
  - Eine Prozessorsimulation des Controllers wird nicht benötigt.
  - Die Modellierung legt den Typ des Controllers nicht fest.

  Dem stehen die folgenden Nachteile gegenüber:
  - Die Portierung wird nur dann mit vertretbarem Aufwand zu bewältigen sein, wenn die Software in einer maschinenunabhängigen Sprache geschrieben worden ist. In der Regel wird dies die Sprache C sein.
  - Die Funktionalität von Hardwarekomponenten des Controllers kann nicht oder nur ansatzweise nachempfunden werden. Dadurch wird das Ablaufverhalten der Software verändert, insbesondere alle Aktionen, die unmittelbar mit Hardwarekomponenten des Controllers in Verbindung stehen, wie z.B. die Reaktion auf Interrupts.

Im Rahmen dieser Arbeit wird letztere Möglichkeit realisiert, da ein geeigneter Simulator auf dem Modellierungsrechner nicht zur Verfügung steht. Von der für das System verfügbaren Software wird zur Demonstration der Machbarkeit der Kernel an das Modell angebunden. Dies erlaubt es, mit dem modellierten System auf die gleiche Art zu kommunizieren wie mit dem realen Mikrosystem. Diese Art der Einbindung von Software in das Modell geht über den objektorientierten Ansatz hinaus, und es muß betont werden, daß die demonstrierte Anbindung der Software nur einen Weg darstellt, die Datenverarbeitung im Rahmen einer Modellierung zu realisieren.

Eine weitere Möglichkeit besteht darin, Teile der Datenverarbeitung als Methoden in das Modell zu integrieren, was exemplarisch für Routinen zur Sensorkalibrierung durchgeführt wird, die der Demonstrationssoftware entnommen sind. Speziell in einer frühen Phase der Modellierung, in der große Teile der Software des realen Systems noch nicht vorliegen, ist dieser Ansatz sinnvoll.

Weitere Überlegungen zur Modellierung von Datenverarbeitung finden sich in der Diskussion am Ende dieses Kapitels.

Informelle Beschreibung
Die informelle Beschreibung der Datenverarbeitung kann, je nach Art ihrer Einbindung in das Modell, verschieden aussehen. Wenn nur einzelne Algorithmen als Methoden modelliert werden, kann die Arbeitsweise des Algorithmus erläutert werden. Wenn große Teile des Software des realen Systems in das Modell eingebunden werden, kann eine Auflistung einzelner Routinen und ihrer Funktion sinnvoll sein.

4.3 Arbeiten mit dem Modellierungswerkzeug

Die Implementierung der Klassen und Objekte geschieht mit den von Nexpert Object zur Verfügung gestellten Editoren. Nach erfolgter Modellierung durch einen aktiven Anwender kann ein passiver Anwender mittels der implementierten Schnittstelle auf das Modell zugreifen. Abbildung 26 zeigt die Schnittstelle in einer Übersicht. Sie erlaubt es dem Benutzer, Klassen und Objekte des Modells aufzulisten und ihre Attribute sowie deren Werte zu inspizieren. Die Beziehungen zwischen Klassen und Objekten (ist-ein, hat-ein, Instanz-von) können mit einem Browser veranschaulicht werden (siehe letztes Kapitel, Abbildung 13).

Die Werte der Attribute können für die Dauer des Aufrufs der Modellierungswerkzeugs durch Überschreiben in der Attributliste modifiziert werden. Beispielsweise kann die Spaltbreite eines Sensors auf diese Weise modifiziert werden, worauf sich auch von ihr abhängige Parameter wie die Kapazität des Sensors ändern.

Für Werte, die sehr häufig oder quasi kontinuierlich geändert werden sollen, können sogenannte "Signalquellen" interaktiv definiert werden. Sie erlauben es, Werte von beliebigen Attributen mit Hilfe von Schiebereglern in einem vom Anwender vorgegebenen Bereich zu verändern. Dies kann dazu verwendet werden, eine externe Anregung von Sensoren zu simulieren. Abbildung 27 zeigt einen solchen Schieberegler.

Eine zusätzliche Hilfe beim Identifizieren von Objekten des Modells gibt der Geometrieviewer. Mit ihm kann die Größe und relative Lage von Objekten angezeigt werden, von denen Geometrieinformationen im Modell abgelegt sind. Abbildung 28 zeigt das Hybridmodul des

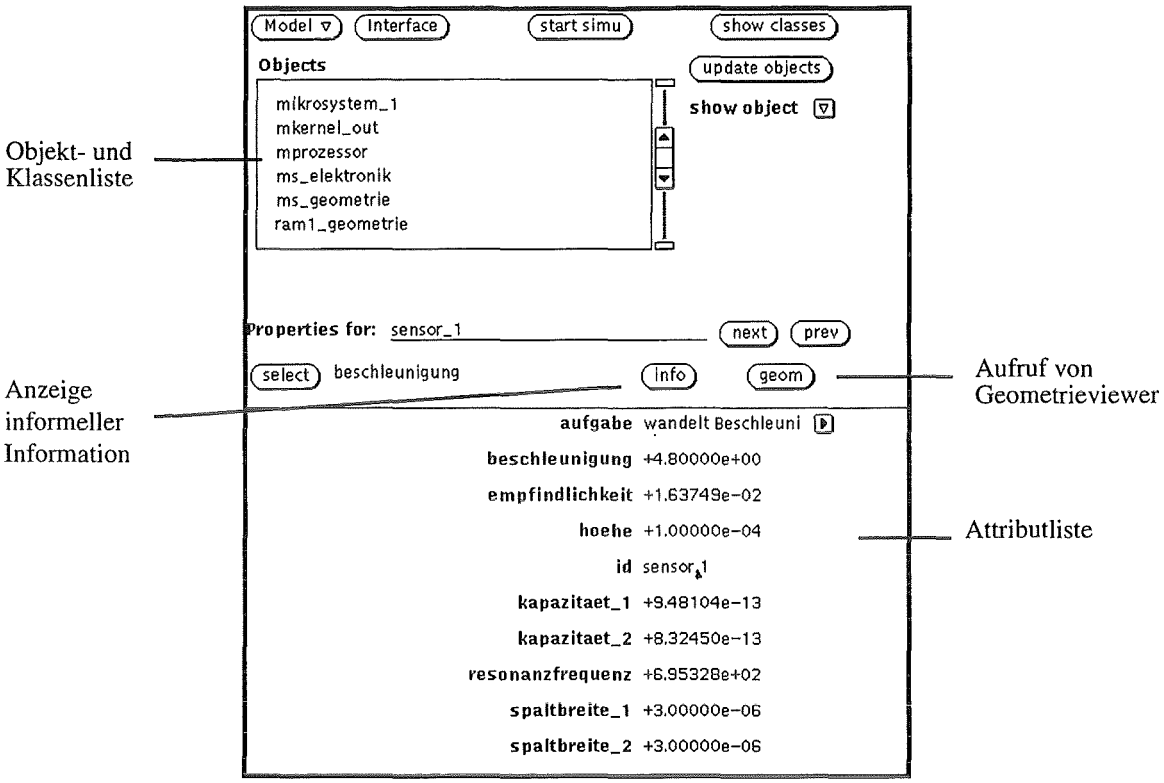


Abb. 26 Schnittstelle für passive Benutzer

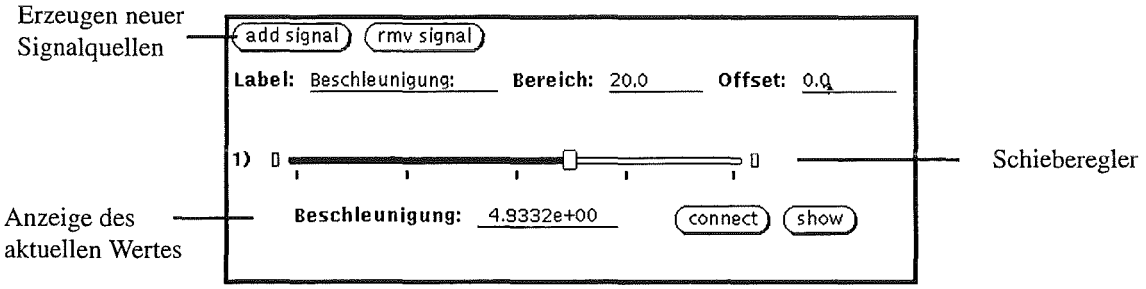


Abb. 27 "Signalquelle" zum Verändern von Attributwerten im Modell. Der gezeigte Regler kann dazu verwendet werden, Sensoren mit Beschleunigungen anzuregen.

Mikrosystems im Geometrieviewer. Über die angezeigten Objekte kann der Benutzer weitere Informationen durch "Anklicken" mit der Maus erhalten. Die Geometrieinformationen geben nicht notwendigerweise die exakte Größe der realen Elemente an, aber man erhält einen Größenvergleich verschiedener Komponenten auf einen Blick. Weiter ermöglicht der Geometrieviewer neben dem Klassen- und Objektbrowser eine weitere Art des Zugriffs auf Elemente des Modells.

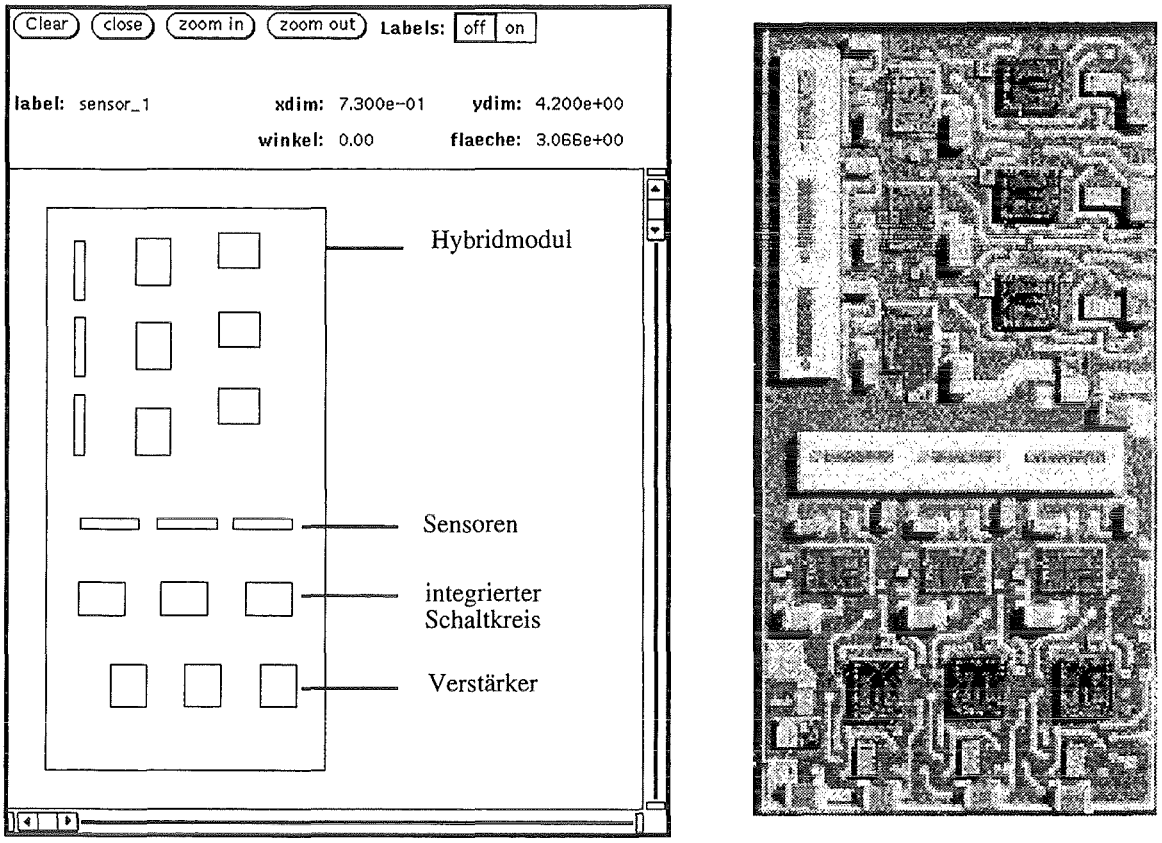


Abb. 28 Schematische Darstellung des Hybridmoduls mit Geometrieviewer. Angezeigt werden Größe und relative Lage der Sensoren, integrierten Schaltkreise und Instrumentenerstärker. Rechts ist zum Vergleich das reale Hybridmodul abgebildet [Gemmeke93].

Die grafische Visualisierung einer funktionalen Abhängigkeit zwischen Attributen des Modells wird ebenfalls zur Verfügung gestellt. Wie bei den "Signalquellen" sind dabei die Attribute nicht vorgegeben, sondern können vom Anwender frei gewählt werden. Abbildung 29 zeigt dies an einem Beispiel.

Diese beiden Hilfsmittel stehen stellvertretend für eine ganze Klasse von Werkzeugen, welche die im Modell enthaltenen Informationen in einer anwendergerechten Form aufbereitet darstellen. Die Anforderungen an solche Werkzeuge hängt auch von den zu modellierenden Mikrosystemen ab. Auf dem Gebiet der Mikrosystemtechnik muß noch mehr Erfahrung gesammelt werden, bevor weitere Werkzeuge dieser Art sinnvoll spezifiziert werden können.

Eine weitergehende informelle Beschreibung ist, falls vorhanden, an die Attribute von Klassen oder Objekten gekoppelt und kann von der Benutzeroberfläche aus aufgerufen werden. Der Inhalt dieser Information hängt von der jeweiligen Modellierung ab. Als Beispiel zeigt Abbildung 30 ergänzende Informationen zum verwendeten Mikrocontroller und eine Fotografie des verwendeten LIGA-Beschleunigungssensors zusammen mit einem integrierten Schaltkreis. An dieser Stelle soll noch einmal auf das schon auf Seite 32 erwähnte Konsistenzproblem hingewiesen werden: das Modellierungswerkzeug kann nur in begrenztem Umfang dazu beitragen, daß der objektorientierte Teil des Modells und die informellen Beschrei-

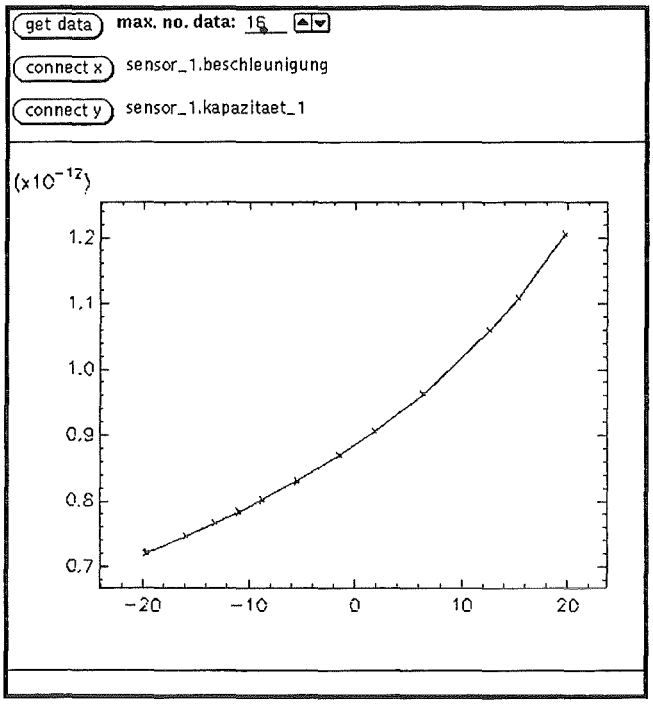


Abb. 29 Funktionaler Zusammenhang zwischen Beschleunigung und einer Kapazität eines Sensors.

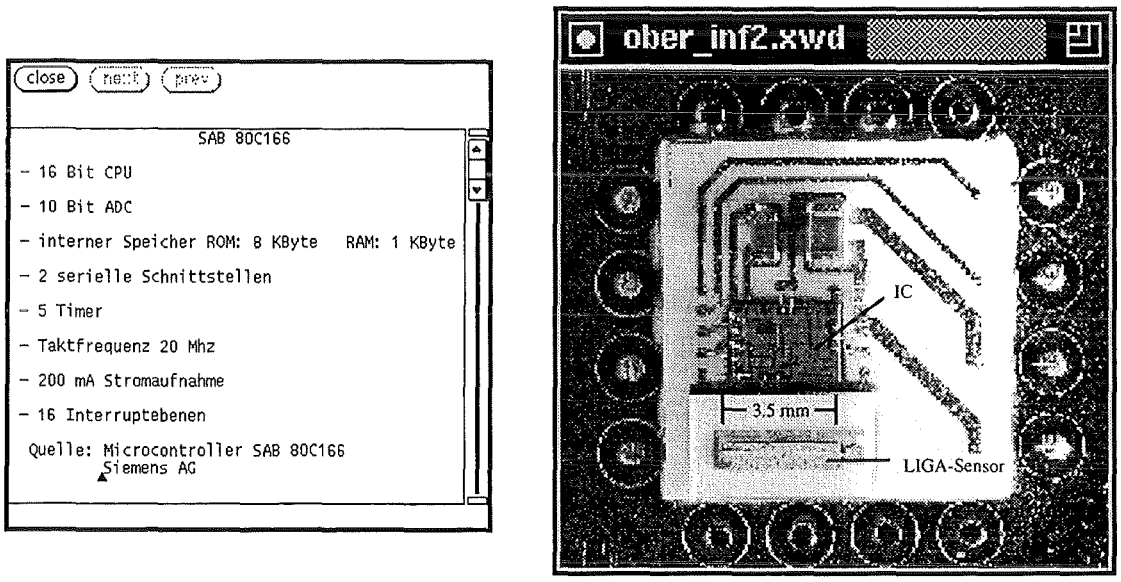


Abb. 30 Informelle Beschreibung in Form eines Datenblattes für den Mikrocontroller und in Form einer Fotografie vom Beschleunigungssensor.

bungen keine widersprüchlichen oder falschen Angaben machen. Da diese Aufgabe vom

Anwender übernommen werden muß, sollten die informellen Beschreibungen sich auf das wirklich Notwendige beschränken, d.h. der Modellierende sollte der Versuchung widerstehen, zu viele Detailinformationen darin unterzubringen.

Bereits bei der Diskussion über die Anbindung der Software an das Modell wurde erwähnt, daß das Modell zusammen mit der Kernel-Software ablaufen kann. Der Anwender kann so mit der Modellierung in vergleichbarer Weise kommunizieren wie mit dem realen System. Im Rahmen dieser Arbeit wurde ein Steuerungsprogramm erstellt, welches auf einer SUN-Workstation abläuft und die Kommunikation mit der Kernel-Software ermöglicht. Im Falle des realen Systems geschieht der bidirektionale Datentransfer über eine serielle RS232-Schnittstelle, beim modellierten System werden die Daten über einen rechnerinternen Kommunikationsmechanismus ausgetauscht. Abbildung 31 zeigt die Benutzeroberfläche des Programms. Mit ih-

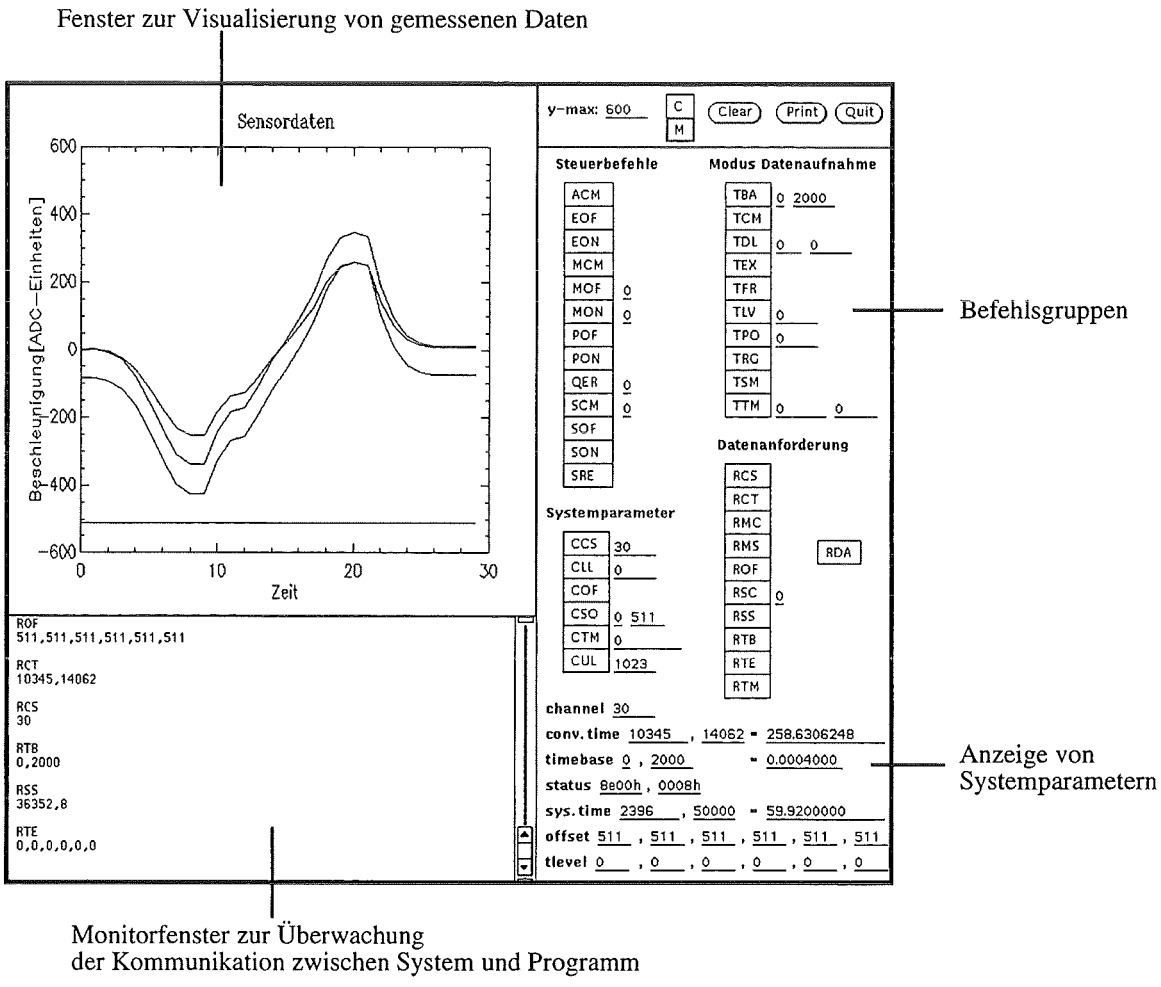


Abb. 31 Benutzeroberfläche des Steuerungsprogramms für das Mikrosystem. Angezeigt werden die "gemessenen" Daten von drei modellierten Sensoren, die der gleichen simulierten Beschleunigung unterworfen sind, aber aufgrund von Variationen der Spaltbreiten der Kondensatoren unterschiedliche Signale liefern.

rer Hilfe können Befehle an das System (real oder modelliert) gesendet werden, gleichzeitig werden vom System kommende Daten in einem Fenster des Programms grafisch dargestellt.

Natürlich bestehen einige prinzipielle Unterschiede zwischen realem und modellierten System: das reale System kann über 1000 Meßwerte pro Sekunde aufzeichnen und verarbeiten, während das Modell um Größenordnungen langsamer ist (einige Meßwerte pro Sekunde oder mehr, abhängig vom eingesetzten Rechner). Eine eingehender Vergleich folgt in der Diskussion. Dennoch kann die prinzipielle Arbeitsweise des realen System gut veranschaulicht werden.

4.4 Diskussion

4.4.1 Vergleich zwischen realem und modelliertem System

Die Qualität einer Modellierung kann in der Regel nicht daran gemessen werden, wie genau ein Modell das reale System wiedergibt. Die Modellierung eines Mikrosystems beginnt ja schon in der Konzeptionsphase und endet zu einem Zeitpunkt, an dem das reale System noch nicht vorliegt. Das primäre Ziel der Modellierung in dieser Phase ist es, die prinzipielle Arbeitsweise und den Aufbau des Systems zu einem frühen Zeitpunkt zu veranschaulichen ("Rapid Prototyping"), wobei Abstriche an der Genauigkeit in Kauf genommen werden können.

Im vorliegenden Fall kann ein Vergleich zwischen modelliertem und realem System stattfinden, da beide annähernd zeitgleich erstellt wurden. Zu diesem Vergleich kann man die statischen und dynamischen Eigenschaften des Gesamtsystems heranziehen.

Vergleich der statischen Eigenschaften
Ein Vergleich ergibt, daß die wesentlichen Komponenten des realen Systems im Modell enthalten sind. Allerdings fällt auf, daß die Modellierung in ihrer Genauigkeit von Komponente zu Komponente stark variiert. Dies ist im Sinn einer Systembeschreibung: von den im System enthaltenen Komponenten werden nur die Eigenschaften modelliert, welche die Gesamtsystemfunktion positiv oder negativ beeinflussen. Daher werden die Beschleunigungssensoren relativ detailliert beschrieben (siehe Abbildung 23), weil hier viele prozeßabhängige Parameter wie Materialkonstanten oder Geometriegrößen direkt in das Meßergebnis eingehen. Durch das Verändern verschiedener Parameter können sowohl verschiedene Varianten modelliert werden, etwa Sensoren mit einer anderen Strukturhöhe, als auch bestimmte, in der Realität vorkommende Fehler simuliert werden. Im Gegensatz dazu kann die Beschreibung der Elektronik auf einem abstrakteren Niveau erfolgen, weil vor allem bei kommerziellen Bauteilen die funktionsbestimmenden Größen durch das Datenblatt vorgegeben sind, etwa bei einem Verstärker der Verstärkungsfaktor. Weitere Angaben machen die Modellierung umfangreicher, ohne für das Gesamtsystem relevant zu sein, daher können und sollten sie unterbleiben. Die Angaben über Platzbedarf und Plazierung sind für Sensoren und elektronische Bauelemente gleichermaßen relevant und vermitteln, grafisch aufbereitet, einen Eindruck vom Systemaufbau und der Sensoranordnung (Abbildung 28).

Ein Vergleich zwischen den Eigenschaften realer Bauelemente und ihrer notwendigerweise abstrakten und unvollständigen Modellierung muß auch an den Absichten gemessen werden, welche beim Entwurf verfolgt werden. Im Kontext einer kommerziellen Produktion von Mikrosystemen könnten beispielsweise die Kosten für einzelne Komponenten und daraus abgeleitet die Kosten für das Gesamtsystem so entscheidende Merkmale sein, daß sie in einer Systemmodellierung nicht fehlen dürfen.

Vergleich der dynamischen Eigenschaften

Beim Vergleich der dynamischen Eigenschaften muß man zwischen dem Verhalten von Komponenten und dem des Gesamtsystems unterscheiden. Den einzelnen Komponenten liegen, wie bereits ausgeführt, idealisierte und vereinfachte Modellbeschreibungen zugrunde. So wird das Verhalten des Beschleunigungssensor durch die Lösung einer Differentialgleichung für ein Feder-Masse-System beschrieben. Die Genauigkeit einer solchen vereinfachten Beschreibung reicht natürlich nicht an die anderer Verfahren wie FEM heran (für einen Vergleich zwischen theoretischen Berechnungen und experimentellen Messungen siehe [Burbaum91]. Dafür läßt sie sich wesentlich schneller auf einem Rechner implementieren und auswerten, was für die Erstellung einer Systemmodellierung unbedingt notwendig ist ([Müller-Glaser94] gibt für die FEM-Simulation eines Beschleunigungssensors Rechenzeiten von mehreren Stunden auf Hochleistungsworkstations an).

Aus der Kopplung der Verhaltensmodelle der einzelnen Komponenten ergibt sich dann das Verhalten des gesamten Modells, welches wiederum das Verhalten des gesamten realen Systems modelliert. Die Genauigkeit der simulierten Meßergebnisse ist notwendigerweise durch die Verwendung vereinfachter Komponentenmodelle beschränkt. Die Modellierung kann dann aber als erfolgreich durchgeführt angesehen werden, wenn sie es ermöglicht, die funktionalen Eigenschaften des Gesamtsystems möglichst früh noch in der Entwurfsphase beurteilen zu können. Dazu gehört vor allem die Interaktion des System mit seiner Umwelt (Anwender, externe Rechner).

Im vorliegenden Fall kommuniziert das reale System über eine RS232-Schnittstelle unter Verwendung eines definierten Protokolls mit einem anderen Rechner. Abbildung 32 zeigt einen Ausschnitt der Kommunikation zwischen realem System und externem Rechner auf der einen Seite, und Modellierung und externem Rechner auf der anderen Seite. Es zeigt sich, daß die Modellierung in vergleichbarer Weise auf die Kommandos des externen Rechners reagiert (Einstellen von Parametern, Start der Messung, Übertragung von Daten etc.). Insoweit erfüllt sie die an sie gestellten Forderungen.

Es wurde bereits darauf hingewiesen, daß die Geschwindigkeit des dynamischen Teils des Modells die des realen System nicht annähernd erreichen kann. Zwar sind Verbesserungen möglich, da ein großer Teil der Geschwindigkeitseinbußen durch Nexpert Object verursacht wird. Hier kann man davon ausgehen, daß durch Verwendung eines anderen Werkzeugs Verbesserungen zu erreichen sind. Auch der Einsatz schnellerer Rechner ist denkbar. Das Mikrosystem zur Messung von Beschleunigungen ist, wie schon erläutert wurde, nicht repräsentativ im Hinblick auf die in Mikrosystemen pro Zeiteinheit anfallende Datenmenge. Bei Systemen mit langsam ansprechenden Sensoren, etwa ChemFET's, ist das Datenaufkommen sehr viel geringer, so daß ein Modell hier die Geschwindigkeit eines realen Systems sogar übertreffen kann. Daher wird in dieser Arbeit darauf verzichtet, besonderen Aufwand für eine Steigerung der Modellgeschwindigkeit zu treiben. Die prinzipielle Funktionsweise des Systems kann auch bei geringem Datendurchsatz durch das Modell gezeigt werden. Negativ macht sich die kleine Datenrate erst dann bemerkbar, wenn die Verarbeitung großer Datenmengen simuliert werden soll.

Im Modell nur unvollkommen wiedergegeben wird das zeitliche Verhalten des Systems. Das Meßsignal des Beschleunigungssensors als schwingendes System hängt bei nichtstatischen Beschleunigungen davon ab, ob diese in der Nähe seiner eigenen Resonanzfrequenz liegen. Auch die elektronischen Schaltkreise haben jeweils eine charakteristische Übertragungsfunk-

a)

SCM 1 ; Messung mit Sensor 1
RCS ; Abfrage der Kanalbreite
10 ; Systemantwort
CCS 100 ; Setzen der Kanalbreite
RCS ; Abfrage der Kanalbreite
100 ; Systemantwort
TBA 7,2000 ; neue Abtastrate
PON ; Starten einer Messung
RSC 1 ; Anforderung Daten von Sensor 1
< Daten des 1. Sensors > ; Systemantwort

b)

SCM 1 ; Messung mit Sensor 1
RCS ; Abfrage der Kanalbreite
10 ; Modellantwort
CCS 100 ; Setzen der Kanalbreite
RCS ; Abfrage der Kanalbreite
100 ; Modellantwrot
TBA 7,2000 ; neue Abtastrate (wird ignoriert)
PON ; Starten einer Messung
RSC 1 ; Anforderung Daten von Sensor 1
< Daten des 1. Sensors > ;Modellantwort

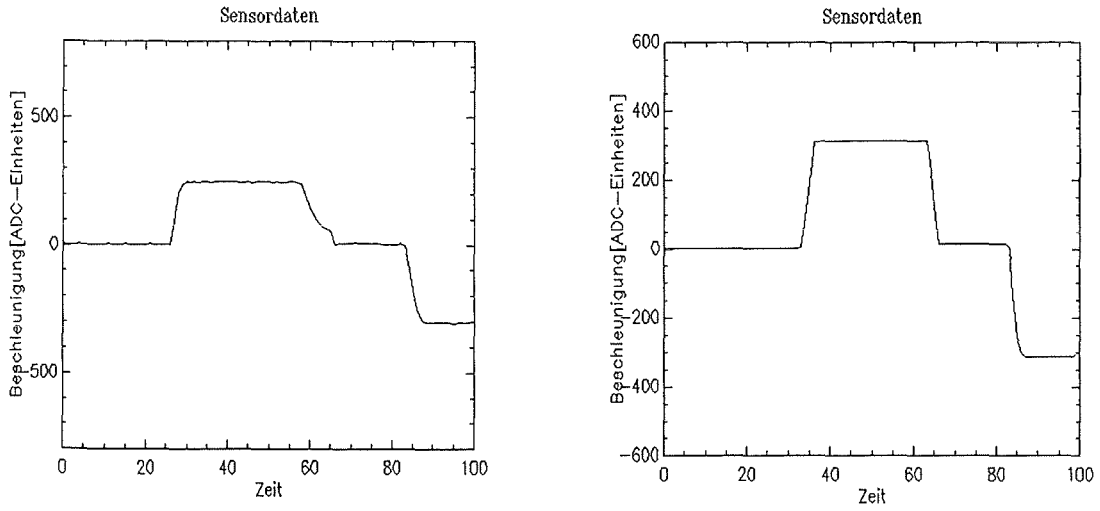


Abb. 32 Kommunikation eines externen Rechners a) mit dem realen System, b) mit der Modellierung. Die Modellierung führt die gleichen Kommandos aus und simuliert so das Verhalten des Gesamtsystems, allerdings nicht in Realzeit.

tion, die von der Frequenz der Eingangssignale abhängt. Eine genauere Modellierung des Sensorverhaltens ist durch eine Zeitintegration seiner Bewegungsgleichung zu erreichen. Dies würde aber einen erheblichen Aufwand bei der Berechnung erfordern. Ein einfacherer Ansatz wird im nächsten Kapitel besprochen.

Das Modell gibt nur mittelbar wieder, daß innerhalb des realen Mikrosystems ein hohes Maß an Parallelität vorhanden ist, d.h. viele Komponenten sind gleichzeitig aktiv. Im Modell werden Beschleunigungen datentechnisch so behandelt, als ob sie sequentiell die einzelnen Komponenten Sensor, integrierter Schaltkreis und Verstärker durchlaufen, während in Wirklichkeit diese Komponenten parallel jeweils für sich ihr Eingangssignal in ein Ausgangssignal transformieren. Ebenso können mehrere Komponenten des Mikrocontrollers gleichzeitig aktiv sein, der ADC kann neue Werte digitalisieren, während der Prozessor noch die alten Daten bearbeitet. Auch auf der Ebene der Datenverarbeitung findet sich Parallelität: obwohl sie auf nur einem Prozessor abläuft, werden Aufgaben wie Auslesen des ADC, Bereitstellen einer

Systemzeit, Datenaufzeichnung und Datenübertragung an ein externes System interruptgesteuert quasiparallel durchgeführt. Das dynamische Modell verrichtet diese Aufgaben sequentiell.

4.4.2 Allgemeine Aspekte einer Modellierung

Das Ergebnis einer Modellierung kann nur mit Einschränkungen dazu verwendet werden, die Auswirkungen eines möglichen Fehlverhaltens des Systems vor seiner Realisierung aufzuzeigen. Dies hat mehrere Ursachen:

- Die vereinfachten Komponentenmodelle im Modell sind nur Näherungen des Verhaltens der realen Komponenten sein.
- Fehler beim Fertigungsprozeß können nicht in jedem Fall in der Entwurfsphase vorausgesagt und in ihren Auswirkungen abgeschätzt werden.
- Bei kommerziellen Komponenten ist man für die Modellierung auf Informationen der mitgelieferten Datenblätter angewiesen, die sich als unzutreffend erweisen können.
- Die Auswirkungen von Defekten in Komponenten können nicht in jedem Fall vorhergesagt werden.

Die Modellierung der Datenverarbeitung wird hauptsächlich durch eine Anbindung der Software des realen Systems an das Modell realisiert. Diese Vorgehensweise ist in mehrfacher Hinsicht ein Spezialfall:

- Die durchgeführte Art der Anbindung ist nur dann einfach möglich, wenn die Software so hardwareunabhängig geschrieben ist, daß sie auf Mikrosystem und Modellierungsrechner gleichermaßen ablauffähig ist.
- Konventionelle Software, die auf einem Prozessor abläuft, ist nur einer von mehreren möglichen Wegen, die Datenverarbeitung in einem Mikrosystem zu realisieren. Einfache Formen der Datenverarbeitung lassen sich auch mit programmierbaren logischen Bausteinen realisieren.

Die Beschreibung der Modellierung des Systems ist damit beendet. Im folgenden Kapitel sollen jetzt Modifikationen am System und ihre Auswirkungen auf die Modellierung unter verschiedenen Gesichtspunkten untersucht werden.

5 Erweiterung des Modells

Eine Veränderung am Modell eines Mikrosystems kann aus unterschiedlichen Gründen erfolgen:

- Wichtige Faktoren, die das Systemverhalten beeinflussen, sind bisher nicht modelliert worden. Dazu gehört auch die Beeinflussung von Komponenten untereinander.
- Die Komponentenentwicklung führt zu einer Verbesserung von vereinfachten Komponentenmodellen, die in das Modell einzubinden sind.
- Eine Änderung der Spezifikationen oder Anforderungen an das zu entwickelnde System macht es notwendig, neue Aspekte bei der Modellierung zu berücksichtigen.
- In der Konzeptionsphase sollen mehrere Varianten eines zu entwickelnden Systems modelliert werden.
- Eine abgeschlossene Modellierung eines realisierten System soll als Ausgangspunkt der Modellierung einer Erweiterung oder Neukonstruktion des Systems dienen.
- Parallel zur Realisierung des Systems kann das Modell detaillierter ausgearbeitet werden.

Je nach Art der Änderungen sind die Auswirkungen auf das Modell unterschiedlicher Art:

- Es können neue Attribute in Klassen definiert werden.
- Neue Objekte von bestehenden Klassen werden erzeugt.
- Es werden von bestehenden Klassen neue Klassen abgeleitet.
- In bestehenden Klassen werden neue Methoden definiert.
- Bisher bestehende Klassen und Objekte werden gelöscht.

Anhand von Erweiterungen des Mikrosystems zur Messung von Beschleunigungen sollen jetzt einige mögliche Modifikationen des Modells vorgestellt werden.

5.1 Modellierung von Störgrößen

Jeder reale Sensor reagiert nicht nur auf seine eigentliche Meßgröße, sondern auch auf störende Einflüsse, die sein Ergebnis verfälschen. Ob sie in eine Modellierung einzubeziehen sind, hängt auch vom Anwendungsbereich ab, d.h. wenn sichergestellt ist, daß bestimmte Störgrößen in der Anwendung nicht auftreten, müssen sie nicht modelliert werden. Ebenso können störende externe Einflüsse durch eine geeignete Kapselung des gesamten Systems oder von einzelnen Sensoren neutralisiert werden. Falls abgeschätzt werden kann, daß die durch eine Störgröße verursachten Meßfehler das eigentliche Ergebnis so wenig beeinflussen, daß die Genauigkeit der Messung nur unwesentlich beeinträchtigt wird, kann eine Modellierung ebenfalls unterbleiben.

Für die im Mikrosystem eingesetzten Sensoren lassen sich zwei verschiedene Störgrößen identifizieren: Querbeschleunigungen und Temperaturschwankungen.

In [Burbaum91] wird die durch eine Querbeschleunigung verursachte Signaländerung abgeschätzt, sie ist für "vernünftige" Sensorgeometrien vernachlässigbar klein. Demnach ist eine Modellierung nicht erforderlich.

Der Einfluß einer Temperaturänderung auf LIGA-Sensoren wird in [Strohrmann92] untersucht. Dabei sind es primär zwei Effekte, welche das Meßergebnis verfälschen. Zum einen ändert sich der Elastizitätsmodul des Sensormaterials mit der Temperatur, zum anderen ändert sich die Sensorgeometrie durch unterschiedliche Ausdehnung der verschiedenen Materialien bei Erwärmung. Da Temperaturschwankungen in vielen Anwendungsbereichen nicht vermeidbar sind, eine Kapselung durch passive Wärmedämmung oder aktive Temperaturregulierung zu aufwendig ist und das Ergebnis des Sensors maßgeblich beeinflußt wird, muß dieser Effekt in das Modell aufgenommen werden.

Zunächst wird die Klasse `LIGA_SENSOR` um ein Attribut erweitert, welches die aktuelle Temperatur des Sensors beschreibt. Es wird nicht in der Klasse `BESCHLEUNIGUNGSSENSOR` eingeführt, da die Temperaturabhängigkeit keine Eigenschaft ist, die für Beschleunigungssensoren allgemein relevant ist. Der Elastizitätsmodul in `MATERIAL_PARAMS` ist jetzt keine Konstante mehr, sondern wird bei einer Temperaturänderung nach folgender Formel modifiziert:

$$E(T) = E_0\,(1 + \alpha_{Em}\,(T - T_0)) \qquad \text{mit} \qquad \begin{aligned} T_0&: \text{ Raumtemperatur} \\ \alpha_{Em}&: \text{ Temperaturkoeffizient des Elastizitätsmoduls (-0.002 \%/K)} \\ E_0&: \text{ Elastizitätsmodul von Nickel } (2.06 \cdot 10^{11}\ \text{Pa}) \end{aligned} \tag{7}$$

was sich ohne Aufwand in Nexpert Object implementieren läßt. Da der Elastizitätsmodul in die Resonanzfrequenz und Empfindlichkeit eingeht, müssen diese auch neu berechnet werden.

Die Verformung von seismischer Masse und Gegenelektrode bei Temperaturänderung bewirkt eine Veränderung der Spaltbreite, wobei die Änderung der Massenbreite den größeren Anteil hat. Dies führt zu einer Kapazitätsänderung. Ebenso führt eine Ausdehnung der gesamten Struktur senkrecht zur Substratebene zu einer Kapazitätsänderung, dieser Effekt ist aber klein gegebenüber der Änderung des Kondensatorspalts. Die Spaltverkleinerung kann im wesentlichen durch die Anwendung folgender Formeln [Strohrmann93b] abgeschätzt werden:

$$\begin{aligned} \Delta d_1 &= -d_M\,\alpha_\Delta\,(T - T_0) \\ \Delta d_2 &= -d_G\,\alpha_{eff}\,(T - T_0) \end{aligned} \qquad \text{mit} \qquad \begin{aligned} d_M&: \text{ Breite seismische Masse} \\ d_G&: \text{ Breite Gegenelektrode} \\ \alpha_\Delta&: \text{ Differenz Ausdehnungskoeffizienten von seismischer Masse und Substrat} \\ \alpha_{eff}&: \text{ effektiver Ausdehnungskoeffizient} \end{aligned} \tag{8}$$

Auch sie sind einfach in ein Modell einzubauen.

Nicht nur die Sensoren, auch die Elektronik ist der Temperaturänderung unterworfen. Die Auswirkungen sind aber vernachlässigbar gegenüber Temperaturempfindlichkeit der Sensoren.

In diesem Zusammenhang läßt sich ein mikrosystemtypischer Effekt diskutieren: störende Einflüsse wirken nicht nur von außen auf das System, die Komponenten beeinflussen sich, bedingt durch die kleinen geometrischen Abmessungen, auch gegenseitig. In diesem Fall werden die Sensoren nicht nur durch eine Änderung der Außentemperatur, sondern auch durch die nicht zu vernachlässigende Wärmeentwicklung in der Elektronik gestört. Die resul-

tierende Temperatur ergibt sich dann aus einer komplexen Überlagerung verschiedener Effekte, zu denen bei einer genauen Betrachtung auch die Geschwindigkeit gehört, mit der die umgebende Luft das System umströmt. Eine genaue mathematische Beschreibung dieses physikalischen Sachverhalts ist viel zu komplex und rechenintensiv, um in eine Systemmodellierung eingebaut zu werden. Hier bietet es sich an, die resultierende Temperatur als externe Signalquelle (analog zu den Beschleunigungen) zu modellieren, welche auf die Kondensatorspaltbreiten und das Elastizitätsmodul des Sensormaterials wirkt.

Die Temperaturabhängigkeit der Sensoren sollte natürlich im Sinne eines Systemgedankens nach außen hin nicht sichtbar werden. Ein Weg, der dies ermöglicht, ist die Messung der Temperatur und anschließende Korrektur der Sensorsignale mit Hilfe der Datenverarbeitung. Dazu muß das Modell um einen Temperatursensor erweitert werden. In der Realität könnte dies ein Sensor sein, der eine Temperatur als analoge Spannung ausgibt, die dann wieder verstärkt und digitalisiert werden kann. Da zunächst die genaue Ausprägung nicht bekannt ist, kann er im Modell als einfaches Objekt erscheinen mit einem Eingang für die Temperatur und einem Ausgang, welcher den gemessenen Wert direkt in einer geeigneten Form an die Datenverarbeitung weitergibt. Dies ist ein Beispiel dafür, wie verschiedene Komponenten eines Systems in einem Modell auf verschiedenen Abstraktionsniveaus beschrieben werden.

5.2 Alternatives Sensordesign

In [Strohrmann93b] wird ein neues Sensordesign (siehe Abbildung 33) vorgestellt, welches das Prinzip des Differentialkondensators beibehält, die freie Formgebung des LIGA-Verfahrens aber zusätzlich ausnutzt, um einige Schwächen der bisher verwendeten Sensoren zu ver-

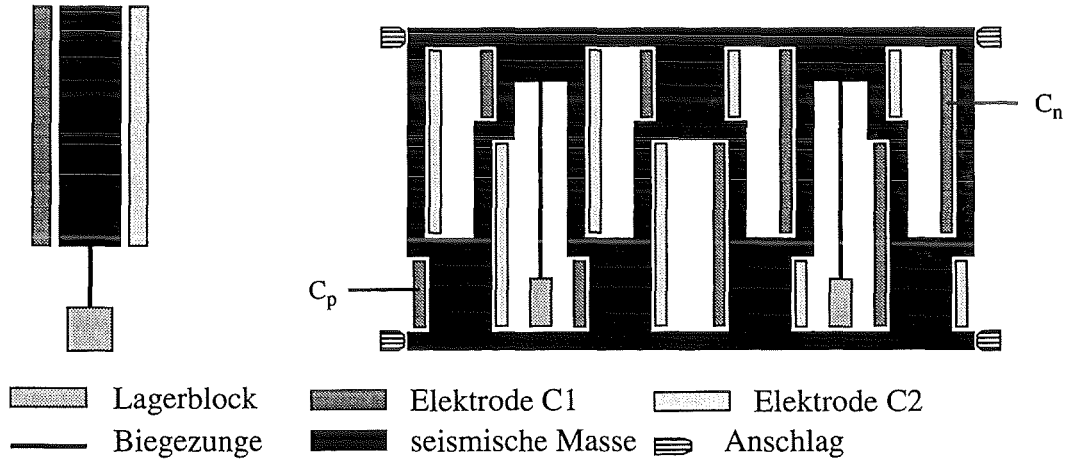


Abb. 33 links: altes Sensordesign, rechts: neues Sensordesign (Zeichnungen nicht maßstabsgerecht). C_p: Kondensator mit positivem Temperaturkoeffizienten, C_n: Kondensator mit negativem Temperaturkoeffizienten.

meiden. Die neuen Sensoren haben u.a. einen wesentlich kleineren Temperaturgang als das alte Design. Für das Modell bedeutet das eine Vereinfachung, da jetzt die Modellierung der Störgröße Temperatur entfällt.

Andererseits ändert sich natürlich die Modellierung des Sensors selber. Die bisherige Modell beinhaltet alle wichtigen geometrischen Größen des Sensors, so daß ein Anwender eine volle Freiheit bei der Wahl der Sensorparameter hat. Dies erscheint bei dem neuen Design nicht mehr sinnvoll zu sein, da hier 16 Kondensatorspalte zu modellieren wären. Deswegen soll die Modellierung so erfolgen, daß sie nur die charakteristischen Größen des neuen Designs enthält.

Dabei ist hilfreich, daß das neue Design eine hohe Symmetrie aufweist. Diese macht es möglich, durch Angabe der Länge von nur zwei Elektroden alle 16 Elektroden zu beschreiben. Alle Kondensatorspalte haben die gleiche Spaltbreite, daher kann die Modellierung durch ein Attribut erfolgen. Während beim alten Design alle geometrischen Größen unabhängig voneinander variiert werden konnten, darf dies jetzt nicht mehr geschehen, da sonst das Design die Eigenschaft der Temperaturkompensation einbüßt oder schlechtere Linearitätseigenschaften aufweist. Das Längenverhältnis der Kondensatoren C_p und C_n muß konstant bleiben. Abbildung 34 zeigt die Klasse `T_KOMP_LIGA_SENSOR` für einen Sensors mit neuem Design,

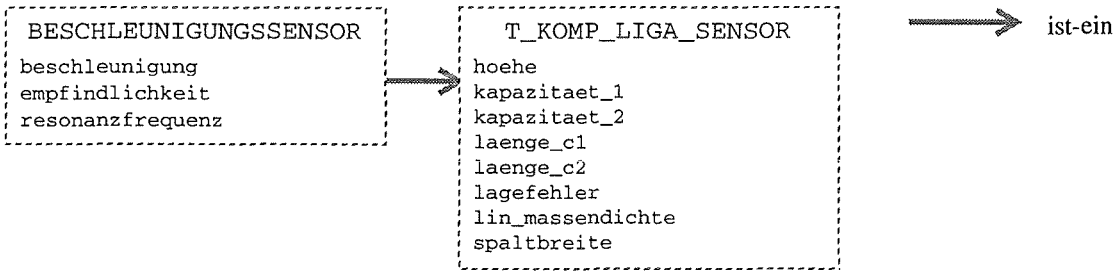


Abb. 34 Klasse `T_KOMP_LIGA_SENSOR` für das neue Sensordesign.

abgeleitet von der Klasse `BESCHLEUNIGUNGSSENSOR`. Die geometrischen Abmessungen der seismischen Massen werden jetzt nicht mehr modelliert. Um trotzdem die für das Sensorverhalten wichtige Masse zu beschreiben, wird eine lineare Massendichte eingeführt, welche die Masse des Sensors pro Strukturhöheneinheit wiedergibt.

Allerdings ist dieses Modell von seinen Möglichkeiten her beschränkt. So kann z.B. nicht die Spaltbreite eines der 16 Kondensatoren oder die Breite von nur einer Biegezunge separat geändert werden, um einen Fehler im Fertigungsprozeß zu simulieren. Ob dies nötig ist, kann nicht generell entschieden werden, sondern hängt von den jeweiligen Gegebenheiten der Fertigung ab. Eine genauere Modellierung der Sensorgeometrie würde es auch gestatten, die verbliebene Temperaturabhängigkeit mit in das Modell aufzunehmen. Dies wird nur dann notwendig, wenn die Genauigkeitsanforderungen an das zu realisierende System hoch sind.

5.3 Alternative Sensorkonfiguration

5.3.1 Änderung der zweidimensionalen Sensorkonfiguration

Das reale Mikrosystem mißt die Beschleunigungen zweier senkrecht zueinander stehender Raumrichtungen. Es sind aber auch andere räumliche Konfigurationen der Sensoren denkbar. In [Menz93a] wird in Zusammenhang mit einer Sensordatenverarbeitung durch neuronale Netze die in Abbildung 35 gezeigte Sensorkonfiguration vorgeschlagen.

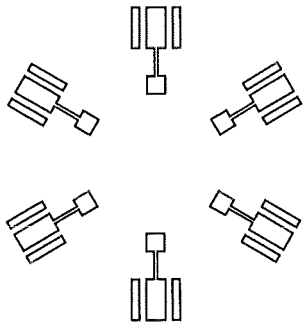


Abb. 35 Beschleunigungssensoren, angeordnet in Form einer Rosette.

Bei der Auswertung von simulierten Beschleunigungen wird die Lage der Sensoren bisher nicht berücksichtigt. Da jeder Sensor durch eine eigene "Signalquelle" angeregt werden kann, ist theoretisch auch die oben gezeigte Sensorkonfiguration simulierbar. Dies läßt einem Anwender zwar mehr Freiheiten, andererseits kann dies auch als Schwäche des Modells interpretiert werden, weil:

- die Beaufschlagung von Sensoren mit unterschiedlichen Signalen eine Sensorkonfiguration simuliert, die den Daten über die geometrische Anordnung der Sensoren widerspricht, d.h. man hat eine Inkonsistenz innerhalb der Modells.
- falls (in der bisherigen Modell) alle Sensoren mit der gleichen Beschleunigung angeregt werden sollen, muß dies der Anwender sicherstellen. Er erhält dabei Unterstützung vom Modellierungswerkzeug, aber nicht durch das Modell selbst. Hier liegt eine mögliche Quelle von Bedienungsfehlern.
- die Sensorkonfiguration als eine grundlegende Systemeigenschaft sehr wahrscheinlich schon zu einem frühen Zeitpunkt der Systementwicklung festgelegt wird, und deshalb eine Flexibilität im Modell keine deutlichen Vorteile bringt.

Diese Nachteile kann man vermeiden, wenn Beschleunigungen so modelliert werden, daß sie auf das ganze Mikrosystem wirken.

Man kann dazu eine Klasse `BESCHLEUNIGUNG` einführen, welche zwei Attribute enthält, mit welchen der Anwender eine gerichtete Beschleunigung in der Ebene simulieren kann (die dritte Raumrichtung muß bei einem zweidimensional messenden System nicht modelliert werden). Man nimmt damit eine auf das Mikrosystem einwirkende Größe explizit in das Modell auf. Von dieser Klasse wird ein Objekt gebildet, welches die im System befindlichen Sensoren als Unterobjekte enthält (siehe Abbildung 36). Damit wird modelliert, daß die Sensoren

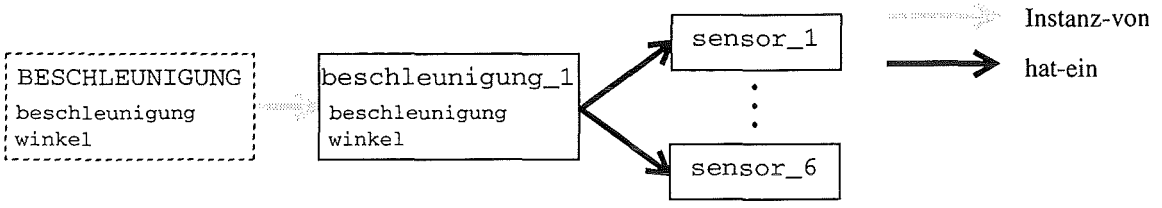


Abb. 36 Modellierung der externen Beschleunigung durch eigene Klasse mit Objekt.

durch die externe Größe Beschleunigung beeinflußt werden. Da die Modellierung der Sensoren schon die Information über ihre Orientierung in der Ebene enthält, kann die resultierende, auf den einzelnen Sensor wirkende Beschleunigung berechnet werden.

An diesem Beispiel kann noch einmal die Freiheit in der Wahl der Modellierung im Zusammenhang mit der unbestimmten Semantik der hat-ein Beziehung diskutiert werden. Die Sensoren sind Unterobjekte zu `beschleunigung_1`, aber das Verhältnis zwischen externer Beschleunigung und den Sensoren, auf die sie wirkt, könnte auch dadurch modelliert werden, daß `beschleunigung_1` gemeinsames Unterobjekt der Sensoren `sensor_1` ... `sensor_6` ist.

Ebenfalls modifiziert werden muß die Datenverarbeitung des Systems. Hierzu können zwei Wege eingeschlagen werden:

- Man behält die Vorgehensweise bei, Software des realen bzw. des zu realisierenden Systems in das Modell einzubinden, wie das bisher der Fall ist. Nachteilig ist, daß diese Software erst neu geschrieben bzw. aus der alten Software heraus entwickelt werden muß. Dies ist zeitaufwendig.
- Man verzichtet auf den vollen Funktionsumfang der Software innerhalb des Modells, sondern implementiert nur einige ausgewählte Algorithmen der Datenverarbeitung als Methoden der Modellierung.

Die Entscheidung darüber hängt vom Umfang der Änderungen ab, die an der alten Software vorgenommen werden müssen.

5.3.2 Erweiterung auf dreidimensionale Messungen

Die Erweiterung des zweidimensional messenden Mikrosystems um Sensoren, die in der dritten Raumrichtung sensitiv sind, kann auf einfache Weise durchgeführt werden, wenn man wieder davon ausgeht, daß die modellierten Sensoren die Richtung der Beschleunigung nicht "kennen". Die Modellierung kann dann einfach um drei zusätzliche Sensoren mit der dazugehörigen Elektronik erweitert werden, die zusätzlichen Sensoren können bei der Simulation durch eine weitere vom Anwender zu definierende "Signalquelle" angeregt werden.

Allerdings ist die Lage der Sensoren in dem Unterobjekt der Klasse `GEOMETRIE_PARAMS` modelliert, welches jedes Sensor-Objekt enthält. Die Angabe eines Winkels und einer zweidimensionalen Positionsangabe reicht jetzt nicht mehr aus, um die Orientierung der Sensoren richtig widerzugeben. Hier kann durch die Ergänzung von Attributen, welche einen zweiten Winkel und eine dritte Raumkoordinate angeben, die Klasse `GEOMETRIE_PARAMS` entsprechend erweitert werden. Ebenso muß jetzt die Ausdehnung der Objekte in der dritten Dimension angegeben werden. Allerdings ist zu überlegen, ob man die zusätzlichen Attribute in `GEOMETRIE_PARAMS` einführt. Das bewirkt, daß die Geometrie aller anderen Gegenstände auch dreidimensional beschrieben wird. Eine Alternative ist, eine neue Klasse `GEOMETRIE_3_PARAMS` zu definieren, welche von `GEOMETRIE_PARAMS` abgeleitet ist und von dieser die Objekte zu instanziieren, welche die Geometrie der Sensoren beschreiben (siehe Abbildung 37).

Dies führt zu einem weiteren Problem. Der im letzten Kapitel gezeigte Geometrieviewer zur Visualisierung der geometrischen Parameter ist jetzt nicht mehr ausreichend, weil er nur zweidimensionale Geometrien darstellen kann. Es ist also nicht nur das Modell selbst, das er-

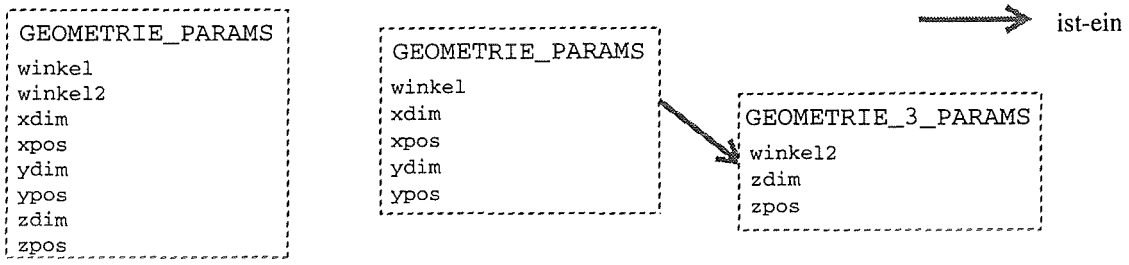


Abb. 37 Links: Erweiterung der Klasse GEOMETRIE_PARAMS. Rechts: Ableitung einer neuen Klasse GEOMETRIE_3_PARAMS.

weitert werden muß, sondern auch das Modellierungswerkzeug. Im Rahmen dieser Arbeit ist eine Implementierung eines dreidimensionalen Geometrieviewers nicht möglich. Es zeigt sich an diesem konkreten Beispiel auch, daß Mikrosysteme manchmal sehr spezielle Hilfsmittel bei der Modellierung erfordern. Dies macht es schwierig, ein umfassendes Modellierungswerkzeug ohne die Kenntnis des zu modellierenden Mikrosystems zu spezifizieren.

Auch hier erfordert die Änderung in der Sensorkonfiguration (Übergang von 2D zu 3D) eine Modifikation der Datenverarbeitung. Die verschiedenen Wege dazu wurden schon im letzten Abschnitt aufgezählt und brauchen daher hier nicht mehr diskutiert zu werden. Wenn die Software stark anwendungsspezifisch geprägt ist, wird man sie unter Umständen weitgehend neu entwickeln müssen, da ein dreidimensional messendes System einen anderen Einsatzbereich haben wird.

5.4 Modellierung von frequenter Anregung und Dämpfung

Das bisherige Modell behandelt nur das Verhalten des Systems bei quasistatischen Beschleunigungen. Bei Anregungen, deren Frequenz in der Nähe der Resonanzfrequenz des Beschleunigungssensors liegt, gibt das Modell die Auslenkung der seismischen Massen und die daraus resultierenden Kapazitätsänderungen nicht mehr richtig wieder. Abgesehen davon können frequente Signale gar nicht dargestellt werden, weil die Zeit bisher gar nicht mitmodelliert wurde (das wurde bereits in der Diskussion im letzten Kapitel angesprochen).

Dennoch gibt es verschiedene Möglichkeiten, das Frequenzverhalten der Sensoren zu modellieren:

- Durch einen erläuternden Text kann darauf hingewiesen werden, daß das Signal des realen Sensors von der Frequenz abhängt, wobei eine grafische Darstellung des Ausgangssignals über der Frequenz eine qualitative Abschätzung ermöglicht.
- Die Übertragungsfunktion kann selbst im Modell berechnet werden, wobei die Frequenz dann als Attribut (z.B. der Beschleunigungssensoren) modelliert wird, die ein Benutzer vorgeben kann, worauf die Kapazitätsänderungen bei anliegender Beschleunigung entsprechend größer oder kleiner ausfallen.

Analog kann auch die Frequenzabhängigkeit anderer Komponenten behandelt werden, falls dies notwendig wird (dies ist auch abhängig vom geplanten Verwendungsgebiet der Sensoren).

Auf ähnliche Weise kann man die Sensordämpfung behandeln. Eine numerische Berechnung der Dämpfung des Sensors durch das ihn umgebende Medium (hier Luft) ist, wenn überhaupt mit ausreichender Genauigkeit möglich, nur mit großem zeitlichem Aufwand durchführbar. Hier muß entweder ein einfaches Modell entwickelt werden [Eberle93], welches eine Abschätzung ermöglicht, oder die Dämpfungskonstante des Sensors als schwingendes System kann aus Messungen bestimmt und dann als Zahlenwert in das Modell eingebracht werden. Das geht natürlich nur, wenn der Sensor als Komponente bereits vorhanden ist, was nicht notwendigerweise der Fall ist.

5.5 Diskussion

In diesem Kapitel wurden verschiedene mögliche Erweiterungen und Modifikationen des bisher realisierten Mikrosystems ihre Auswirkungen auf das Modell behandelt.

Die Änderungen am Modell sind dabei unterschiedlicher Art. Das Einführen von neuen Attributen oder das Erzeugen von neuen Klassen, oft durch Ableitung von bestehenden Klassen, ist ohne größere Probleme möglich, da das bisher bestehende Modell davon unberührt bleibt. Schwieriger ist das Entfernen bereits existierender Klassen und/oder Objekte, da hiervon auch andere Teile des Modells betroffen sein können. Dies gilt auch für die Einführung neuer Klassen im oberen Teil der Vererbungshierarchie. Es ist dabei hilfreich, wenn das Modellierungswerkzeug den Anwender unterstützt, d.h. wenn es z.B. beim Löschen eines Attributs zeigt, welche Teile des Modells betroffen sind. Generell wird eine Erweiterung eines bestehenden Modells um so schwieriger sein, je umfangreicher und detaillierter sie ausgearbeitet wurde. Für eine Erweiterung kann es deshalb sinnvoll sein, nicht die letzte Version eines existierenden Modells als Ausgangspunkt zu nehmen, sondern eine frühere Version zu verwenden (die Notwendigkeit einer Unterstützung der Verwaltung von Versionen wurde schon durch Forderung 14 (Seite 15) begründet).

Eine Erweiterung des Modells, die aus verschiedenen Gründen notwendig werden kann, sollte bereits bei der Ersterstellung eingeplant und bei der Wahl von Klassen bzw. Objekten mit berücksichtigt werden, damit spätere umfangreiche Modifikationen an Klassen- und Objekthierarchien vermieden werden. Die Qualität einer Modellierung und eines Modellierungswerkzeugs muß auch daran gemessen werden, wie leicht Erweiterungen möglich sind. Dies gilt besonders unter dem Gesichtspunkt, daß die Ersterstellung und die Weiterentwicklung des Modells möglicherweise nicht von der gleichen Person durchgeführt werden.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurde gezeigt, wie eine Modellierung von Mikrosystemen durchgeführt werden kann. Dazu wurde eine Methode entwickelt, die statischen und dynamischen Eigenschaften von Mikrosystemen während der Entwicklung auf der Systemebene rechnergestützt zu beschreiben. Damit ist sowohl eine Dokumentation des Systems verbunden als auch eine Simulation des Verhaltens des Gesamtsystems zu einem möglichst frühen Zeitpunkt. Auf der Grundlage dieser Methode wurde ein Modellierungswerkzeug implementiert. Methode und Werkzeug wurden dabei für die Modellierung von Mikrosystemen allgemein ausgelegt. Die Anwendbarkeit von Methode und Werkzeug wurden durch die Modellierung eines konkreten Mikrosystems nachgewiesen. Dabei wurde für das reale System die Datenverarbeitung implementiert und eine Demonstrationsumgebung erstellt.

Da sich Mikrosysteme aus sehr unterschiedlichen Komponenten zusammensetzen (Sensoren, Aktoren, analoge Elektronik, digitale Elektronik, Software), war es notwendig, für ihre Modellierung eine Methode zu wählen, die nicht schon aufgrund inhärenter Beschränkungen nur für die Beschreibung von Teilbereichen verwendbar ist. Die Methode muß für die Beschreibung statischer und dynamischer Aspekte von Mikrosystemen geeignet sein. Gleichzeitig muß sie von kommerziell erhältlichen Tools unterstützt werden. Es wurde gezeigt, daß die Methode der objektorientierten Modellierung, ergänzt um die Integration von informellen Beschreibungen, diese Anforderungen erfüllt.

Um eine Modellierung praktisch durchführen zu können, war es notwendig, ein rechnergestütztes Werkzeug zu spezifizieren und wichtige Teile davon prototypisch zu implementieren. Als Ausgangspunkt wurde dazu die Expertensystemshell Nexpert Object ausgewählt, da sie die Methode der objektorientierten Modellierung unterstützt und bereits über eine grafische Oberfläche zum Erstellen des Modells verfügt. Die für eine geeignete grafische Darstellung von Daten und für einen Zugriff auf das Modell erforderlichen Ergänzungen wurden auf der Basis von Standardsoftware implementiert.

Danach wurde anhand der Modellierung eines Mikrosystems zur Messung von Beschleunigungen, dessen Software ebenfalls im Rahmen dieser Arbeit entwickelt wurde, die Praxistauglichkeit der Methode nachgewiesen. Gemäß der Methode der objektorientierten Beschreibung wurde für die statische Beschreibung eine Klassenhierarchie für das Mikrosystem aufgestellt, so daß einzelne Komponenten des Systems als Objekte modelliert werden konnten. Die dynamische Beschreibung geschieht mit vereinfachten Komponentenmodellen, die teilweise in der Sprache C implementiert wurden. Sie wurden als Methoden in das Modell eingebunden und ermöglichen eine Simulation des Verhaltens der realen Komponenten.

Für die Modellierung der Datenverarbeitung wurden verschiedene Vorgehensweisen diskutiert. Umgesetzt wurde die Einbindung der Software des realen Systems in das Modell. Auf diese Weise kann das Modell einen großen Teil des Verhaltens des realen Systems simulieren, insbesondere die Kommunikation mit einem externen Rechner, die ebenfalls im Rahmen dieser Arbeit implementiert wurde. Daneben wurde ein Algorithmus zur Sensorkorrektur als Methode in das Modell eingebunden.

Das Modell wurde durch einen Vergleich mit dem realen System validiert. Es zeigt sich, daß wichtige Aspekte des Verhaltens des realen Systems durch das Modell wiedergegeben werden. Dies umfaßt sowohl das korrekte Verhalten, wie es aus dem Design zu erwarten ist, als auch bei einem realen System typischerweise auftretende Fehler, die ein nicht korrektes Verhalten des Systems bewirken (siehe Abbildung 31).

Weiterhin wurde durch eine Modifikation in verschiedenen Teilbereichen (Störgrößen, alternatives Sensordesign, andere Sensorkonfiguration) die Flexibilität beim Erstellen einer Modellierung nachgewiesen. Dies ist wichtig, weil zu erwarten ist, daß in der Praxis oft vom Design her ähnliche Mikrosysteme zu entwickeln sind, welche dann in weiten Teilen gleich modelliert werden können.

Damit wurde die in der Einleitung formulierte Aufgabenstellung gelöst.

6.2 Ausblick

Das Erstellen einer Modellierung ist in die Gesamtthematik Unterstützung des Entwurfs von Mikrosystemen eingebettet. Betrachtet man das in [BMFT93] angegebene Schema zum Entwurfsablauf für Mikrosysteme (siehe Abbildung 38), so kann ein Modell von der Spezifikationsphase bis zur Systemrealisierung kontinuierlich weiterentwickelt werden. Dies wird durch die in den Klassenhierarchien manifestierte Abstraktion auf verschiedenen Ebenen erleichtert. Das Modellierungswerkzeug kann und soll allerdings nicht die Aufgaben der in den einzelnen Entwurfsphasen notwendigen speziellen Werkzeuge ersetzen. So bietet es keine Unterstützung bei der Systempartitionierung (d.h. der Aufteilung der verschiedenen Systemfunktionen auf die Komponenten) an, es kann nur helfen, diese Aufteilung zu beschreiben.

Eine Hauptschwierigkeit bei der Erstellung einer Entwurfsunterstützung für Mikrosysteme ist die große Heterogenität der Probleme, denen sich ein Entwickler gegenübersieht. Praktisch bei jedem neu zu entwickelnden System werden, zumindest im jetzigen frühen Stadium der Mikrosystemtechnik, Problemstellungen auftauchen, die vorher in dieser Weise so noch nicht dagewesen sind, und daher auch von keinem existierenden Werkzeug abgedeckt werden. Dies betrifft zwar in erster Linie die Komponenten, hat aber auch Auswirkungen auf das System als Ganzes. Die in dieser Arbeit durchgeführte Modellierung hat aber gezeigt, daß auch sehr heterogene Komponenten durch den objektorientierten Ansatz beschrieben werden können. Daher darf man hoffen, daß er auch bei der Modellierung neuentwickelter Komponenten erfolgreich angewendet werden kann.

Bei einer Modellierung sind wichtige Kriterien die Geschwindigkeit und Leichtigkeit, mit der das Modell erstellt werden kann[1]. Diese wiederum sind stark abhängig von der Qualität und Benutzerfreundlichkeit des Modellierungswerkzeugs. Bei Bewertung der Benutzerfreundlichkeit können zwei Aspekte unterschieden werden:

- Die Erstellung des Modells selber muß leicht möglich sein. Dies kann dadurch geschehen, daß für gewisse Kategorien von Mikrosystemen entsprechende Klassen für die Modellierung schon bereitgestellt werden, z.B. Klassen für verschiedene Arten von Sensoren. Das Zusammensetzen von Objekten dieser Klassen

1. Schließlich soll die Modellierung bei der Lösung von Problemen helfen, nicht selbst eines sein.

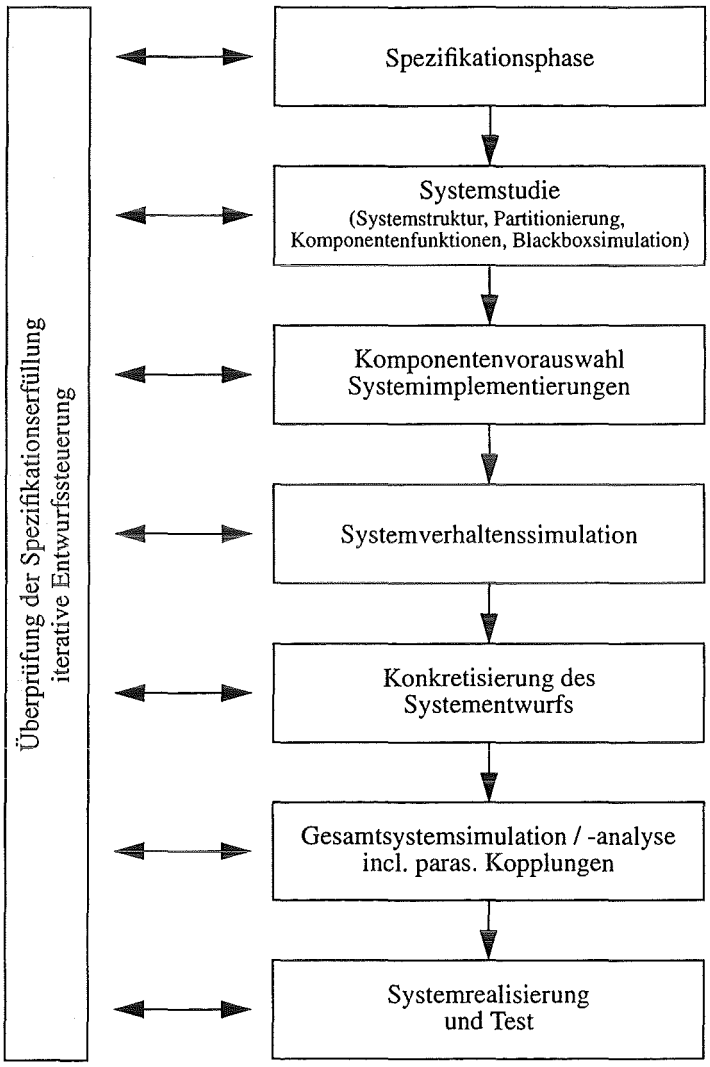


Abb. 38 Entwurfsablauf für Mikrosysteme aus [BMFT93].

sollte dann möglichst interaktiv erfolgen. Ebenso sollte das Definieren von Methoden, welche die Dynamik des Systems beschreiben, vom Modellierungswerkzeug unterstützt werden.

- Die Benutzung des Modells muß erleichtert werden. Dem Dokumentationsaspekt des Modells kann dadurch Rechnung getragen werden, daß das in ihm enthaltene Wissen über das System in geeigneter Weise dem Benutzer präsentiert wird, beispielsweise durch die Darstellung von funktionalen Zusammenhängen in Form von Graphen. Die Simulation kann durch eine geeignete Auswahl von Signalquellen, mit denen Parameter des Modells variiert werden können, erleichtert werden.

Im Rahmen dieser Arbeit war hier nur eine ansatzweise Lösung möglich, da die Erstellung von grafischen Oberflächen zwar keine prinzipiellen Probleme aufwirft, aber sehr zeitintensiv ist.

Eine Erleichterung beim Modellieren ist dann zu erwarten, wenn auf bereits existierenden Arbeiten aufgebaut werden kann. Dies können alte Modelle oder auch nur die Definition von Klassenhierarchien sein, deren Entwurf übernommen werden kann. Allerdings wird auch hier die dynamische Entwicklung der Mikrosystemtechnik dazu führen, daß neu entwickelte Komponenten beschrieben werden müssen und daher oft eine Modellierung von Grund auf durchgeführt werden muß.

In dieser Arbeit wurde ein starkes Gewicht auf die für eine Anwendung wichtigen Gesichtspunkte eines Modellierungswerkzeugs gelegt. Insbesondere die Notwendigkeit der Integration mit anderen Werkzeugen wurde dabei betont. Generell ist die Integration verschiedener Werkzeuge ein noch nicht befriedigend gelöstes Problem. Ansätze zur Lösung sind unter dem Schlagwort "Frameworks" bekannt. In [Gröning93][CFI92] wird eine solche Framework-Software vorgestellt, die eine objektorientierte Datenbasis als Grundlage der zu integrierenden Werkzeuge verwendet. Man kann daher erwarten, daß sich ein objektorientiertes Modellierungswerkzeug in ein solches Framework leicht integrieren läßt. Neben der Integration mit anderen Werkzeugen hätte dies den Vorteil, daß Teile der Benutzerschnittstelle des Frameworks auch für die Modellierung verwendet werden können.

Aufbauend auf den Erfahrungen, die in dieser Arbeit gemacht wurden, ergeben sich folgende Möglichkeiten der Weiterentwicklung:

- Die Eignung von Methode und Werkzeug zur Modellierung sollte an einem weiteren Mikrosystem nachgewiesen werden. Vorzugsweise sollte dies dahingehend ausgesucht werden, daß es andere Komponenten (z.B. Aktoren) enthält als das Mikrosystem zur Messung von Beschleunigungen und auch eine höhere Komplexität aufweist, was Art und Zahl der Komponenten angeht. Damit kann die Flexibilität der Methode getestet und weitere Anregungen zur Ergänzung des Werkzeugs erhalten werden.
- Die Integration mit anderen Werkzeugen im Rahmen eines Frameworks muß konkret untersucht werden.
- Die als Grundlage des Werkzeugs verwendete Version von Nexpert Object ist vor einigen Monaten durch eine neue Version abgelöst worden. Diese muß daraufhin überprüft werden, ob sie Schwächen und Defizite des bisherigen Version vermeidet.
- Unter Berücksichtigung aktueller Entwicklungen im Bereich der OODBS und Frameworks ist zu prüfen, ob statt Nexpert Object ein anderes kommerzielles Programm als Grundlage eines Modellierungswerkzeuges verwendet werden sollte. Prinzipiell ist dies möglich, da spezielle Eigenschaften von Nexpert Object nicht verwendet wurden.

7 Literaturverzeichnis

[Atkinson89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik: *The Object-Oriented Database System Manifesto*, Proc. 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, 1989, pp. 40 - 57

[Becker85] E.W. Becker, W. Ehrfeld, P. Hagmann, A. Maner, D. Münchmeyer: *Herstellung von Mikrostrukturen mit großem Aspektverhältnis und großer Strukturhöhe durch Röntgentiefenlithographie mit Synchrotronstrahlung, Galvanoformung und Kunststoffabformung (LIGA-Verfahren)*, KfK-Bericht 3995, November 1985

[BMFT92] BMFT-Verbundprojekt: *Untersuchung zum Entwurf von Mikrosystemen*, Förderkennzeichen 13 MV 0157, 1. Statusbericht, Mai 1992, S. 5

[BMFT93] BMFT-Verbundprojekt: *Untersuchung zum Entwurf von Mikrosystemen*, Zwischenbericht zum Statusseminar am 5. Oktober 1993

[Booch91] G. Booch: *Object Oriented Design with Applications*, The Benjamin/ Cummings Publishing Company, 1991

[Bortolazzi93a] J. Bortolazzi, J. Ernst, Y. Tanurhan, K. Müller-Glaser: *Rechnergestützte Anforderungsspezifikation für den Mikrosystementwurf*, Workshop "Informationstechnik für Mikrosysteme", Kernforschungszentrum Karlsruhe, Januar 1993, S. 51 - 60

[Bortolazzi93b] J. Bortolazzi: *Prototyp zur Erfassung von Anforderungsspezifikationen*, 1. Statusseminar zum Verbundprojekt METEOR, 6.-7. Oktober 1993, Kernforschungszentrum Karlsruhe

[Brachman83] R.J. Brachman: *What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks*, IEEE Computer, vol. 16 (10), October 1983, pp. 30 - 36

[Brauch92] I. Brauch, H. Eggert, K.P. Scherer, P. Stiller: *Einsatz wissensbasierter Methoden für Konstruktion, Fertigung und Test von LIGA-Mikrostrukturen*, "Information als Produktionsfaktor", 22. GI-Jahrestagung , 28.9. - 2.10. 1992, S. 714 - 721

[Brauch94] I.H. Brauch: *Wissensbasierte Modellierung des LIGA-Fertigungsprozesses*, KfK-Bericht 5238, Dezember 1994

[Brignell89] J.E. Brignell: *Smart Sensors*, in: Sensors - A Comprehensive Survey, vol 1., VCH Verlagsgesellschaft, 1989

[Budd91] T. Budd: *An Introduction to Object-Oriented Programming*, Addison-Wesley Publishing Company, 1991

[Burbaum91] C. Burbaum, J. Mohr: *Herstellung eines mikromechanischen Beschleunigungssensors in LIGA-Technik*, KfK-Bericht 4859, April 1991

[CFI92] CAD Framework Initiative, Inc.: *Design Representation Programming Interface: Electrical Connectivity*, Standards for Electronic Design Automation Release 1.0, 1992

[Chen76] P.P. Chen: *The entity-relationship model - towards a unified view of data*, ACM Transactions on Database Systems, vol. 1, no. 1, March 1976, pp 9 - 36

[Eberle93] F. Eberle: unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, November 1993

[Eggert94] H. Eggert, K.P. Scherer, P. Stiller: *Geometriewissensbasis als Schnittstelle MCAD/ECAD*, 2. Statusseminar zum BMFT-Verbundprojekt METEOR, November 1994, S. 94 - 101

[Eick93] C.F. Eick, B. Czejdo: *Reactive rules for C++*, Journal of Object-Oriented Programming, October 1993

[Fromhein93] O. Fromhein, T. Kühner: unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, Mai 1993

[Gemmeke93] H. Gemmeke, M. Balzer, O. Fromhein, O. Krömer, T. Kühner: *Schaltungsentwurf für Mikrosysteme*, KfK-Bericht 5238, September 1993, S. 170 - 175

[Grabowski93] H. Grabowski, R. Anderl, A. Polly: *Integriertes Produktmodell*, Beuth Verlag, 1993

[Graham93] I. Graham: *Migration using SOMA: A semantically rich method of object-oriented analysis*, Journal of Object-Oriented Programming, February 1993, pp. 31 - 42

[Gröning93] K. Gröning: *Werkzeugintegration mit der Framework-Basissoftware*, 1. Statusseminar zum Verbundprojekt METEOR, 6.-7. Oktober 1993, Kernforschungszentrum Karlsruhe

[Harel87] D. Harel: *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming 8 (1987), pp. 231 - 274

[Heuer92] A. Heuer: *Objektorientierte Datenbanken - Konzepte, Modelle, Systeme*, Addison-Wesley Publishing Company, 1992

[Heytens89] M. L. Heytens, R. S. Nikhil: *GESTALT: An Expressive Database Programming System*, SIGMOD RECORD, Vol. 18, No. 1, March 1989, pp. 54 - 67

[Hoffmann93] W. Hoffmann, H. Eggert, W. Schomburg, D. Seidel: *Elektrochemisches Mikroanalysensystem (ELMAS) für die Ionometrie von Flüssigkeiten*, KfK-Bericht 5238, September 1993, S. 87 - 91

[Kroy89] M. Kroy: *Mikrosystemtechnik unter besonderer Berücksichtigung der Mikromechanik*, AMA Seminar Mikromechanik, Oktober 1989, S. 289 - 298

[Kuhlen91] R. Kuhlen: *Hypertext*, Springer Verlag, 1991

[Lee93] H.J. Lee, W.T. Tsai: *A new partila inheritance mechanism and its applications*, Journal of Object-Oriented Programming, July-August 1993, pp. 53 - 63

[Leuthold90] H. Leuthold, F. Rudolf: *An ASIC for High-resolution Capacitive Microaccelerometers*, Sensors and Actuators, A21-A23(1990), pp. 278 - 281

[Li91] Xiaofeng Li: *What's so bad about rule-based programming?*, IEEE Software, September 1991, pp. 103 - 105

[Lindemann91] K. Lindemann, B. Bürg, H. Eggert: unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, November 1991

[Lindemann94] K. Lindemann, H. Eggert, W. Süß: *Informationstechnik für ein Mikrosystem zur Messung von Beschleunigungen*, 2. Statusseminar zum BMFT-Verbundprojekt METEOR, S. 172 - 178, November 1994

[Manoli91] Y. Manoli, W. Mokwa: *Der intelligente Herzkatheter*, Elektronik 24/ 1991, S. 94 - 100

[Menz91] W. Menz, W. Bacher, M. Harmening, A. Michel: *The LIGA Technique - a Novel Concept for Microstructures and the Combintion with Si-Technologies by Injection Molding*, Proc. MEMS 1991, Nara, Japan, Jan. 30th - Feb. 2nd 1991, pp. 69 - 73

[Menz93a] W. Menz, P. Bley: *Mikrosystemtechnik für Ingenieure*, VCH Verlagsgesellschaft mbH, 1993

[Menz93b] W. Menz: *Die LIGA-Technik und ihr Potential für die industrielle Anwendung*, KfK-Bericht 5238, September 1993, S. 19 - 28

[Meyer90] B. Meyer: *Objektorientierte Softwareentwicklung*, Carl Hanser Verlag, 1990

[Müller-Glaser94] K.D. Müller-Glaser: *Die goldene Mitte*, Elektronik 21/1994 S. 84 - 96 und Elektronik 23/1994 S. 114 - 122

[Mohr94] J. Mohr, M. Strohrmann, O. Fromhein, K. Lindemann: *Ein Beschleunigungssensorsystem in LIGA-Technik - Aufbau und Eigenschaften*, Spektrum der Wissenschaft, Februar 1994, S. 99 - 106

[Nexpert91] *NEXPERT OBJECT Functional Description*, Neuron Data, Inc., January 1991

[Petersen82] Kurt E. Petersen, *Silicon as a Mechanical Material*, Proc. IEEE, vol. 70, no. 5, May 1982, pp. 420 - 457

[Polla86] D.L. Polla, R.S. Muller, R.M. White: *Integrated Multisensor Chip*, IEEE Electron Device Lett., vol. EDL-7, no. 4, April 1986, pp. 254 - 256

[Rauch89] H. Rauch, K.D. Müller-Glaser: *Entwurfsumgebung für Mikrosysteme*, AMA Seminar Mikromechanik, Oktober 1989, S. 1 - 22

[Reichl89] H. Reichl: *Anforderungen an die Aufbau- und Verbindungstechniken für Mikrosysteme*, AMA Seminar Mikromechanik, Oktober 1989, S. 301 - 316

[Reichl93] H. Reichl, M. Kasper: *Strategien und erste Lösungswege für die Systemintegration*, Zwischenbericht zum Statusseminar des BMFT-Verbundprojekts "Untersuchung zum Entwurf von Mikrosystemen", 5. Oktober 1993, Kernforschungszentrum Karlsruhe

[Rembold87] U. Rembold: *Einführung in die Informatik für Naturwissenschaftler und Ingenieure*, Carl Hanser Verlag, 1987

[Riethmüller89] W. Riethmüller: *Möglichkeiten der monolithischen Integration von mikroelektronischen und mikromechanischen Funktionselementen*, AMA Seminar Mikromechanik, März 1989, S. 313 - 329

[Roylance79] Lynn Michelle Roylance, James B. Angell: *A Batch-Fabricated Silicon Accelerometer*, Trans. Electron Devices, vol. ED-26, no. 12, December 1979, pp. 1911 - 1917

[Schuch92] B. Schuch, G. Leicht: *Multichip-Module - ein Weg für kostenoptimierte Lösungen*, Elektronik 1/1992, S. 78 - 83

[Senturia86] S.D. Senturia, R.L. Smith: *Microsensor Packaging and System Partitioning*, 14th Automotive Materials Conference, Ann Arbor, MI, November 19, 1986, pp. 185 - 191

[Senturia92] S.D. Senturia, R.M. Harris, B.P. Johnson, S. Kim, K. Nabors, M.A. Shulman, J.K. White: *A Computer-Aided Design System for Microelectromechanical System (MEMCAD)*, Journal of Microelectromechanical Systems, vol. 1, no. 1. March 1992, pp. 3 - 13

[Shapiro87] S.C. Shapiro, D. Eckroth: *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, 1987

[Shaw93] N. Shaw: *Interfacing Technology for Manufacturing Industry: From Islands of Automation to Continents of Standardization and Beyond*, Proceedings of the IFIP TC5/WG5.10 Working Conference on Interfaces in Industrial Systems for Production and Engineering, Darmstadt, März 1993, pp. 1 - 11

[Siemens90] *Microcontroller SAB 80C166 User's Manual*, Siemens, 1990

[Specht91] D. Specht, M. Forkel, Th. Göbler: *Semantische Modellierung von technischen Objekten und Handlungen*, Tagung "Erfolgreiche Anwendung wissensbasierter Systeme in Entwicklung und Konstruktion", VDI Berichte 903, 1991, S. 145 - 161

[Strohrmann92] M. Strohrmann, J. Mohr: unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, März 1992

[Strohrmann93a] M. Strohrmann, O. Fromhein, W. Keller, K. Lindemann, J. Mohr: *LIGA-Sensoren und intelligente Sensorsysteme zur Messung von Beschleunigungen*, KfK-Bericht 5238, September 1993, S. 65 - 70

[Strohrmann93b] M. Strohrmann: unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, Oktober 1993

[Stroustrup91] B. Stroustrup: *The C++ Programming Language*, Addison-Wesley Publishing Company, 1991

[Süß93a] W. Süß, K. Lindemann: unveröffentlichter Bericht des Kernforschungszentrums Karlsruhe, März 1993

[Süß93b] W. Süß, H. Eggert, M. Gorges-Schleuter, W. Jakob, W. Hoffmann: *Modeling of a Microsystem for the Detection of Ions in Fluids with STATEMATE*, Second Annual International User Group Conference i-Logix, Burlington, 1993

[Tanimoto90] S.L. Tanimoto: *KI: Die Grundlagen*, R. Oldenbourg Verlag, 1990

[Turner91] S.R. Turner: *Nexpert Object*, IEEE Expert, December 1991

[Wallrabe92] U. Wallrabe, P. Bley, J. Mohr: *Entwicklung, Optimierung und Test von elektrostatischen Mikromotoren nach dem LIGA-Verfahren*, KfK-Bericht 5088, September 1992

[Wieder92] A.W. Wieder: *"Systems on Chips": Die Herausforderung der nächsten 20 Jahre*, Informationstechnik 34, 1992, S. 202 - 208

[Wieland95] P. Wieland, C. Düpmeier, H. Eggert, K.P. Scherer: *Entwurf eines Systems zur Erfassung und Weiterverarbeitung von Produktinformation bei der Herstellung von Mikrostrukturen*, FZKA-Bericht 5597, Juli 1995

Anhang

Die Mikrosystemtechnik ist durch eine enge Zusammenarbeit mehrerer Disziplinen gekennzeichnet, und so ist es nicht zu vermeiden, daß Fachbegriffe und gebräuchliche Abkürzungen aus den verschiedenen Gebieten in dieser Arbeit verwendet werden. In einem Glossar werden sie zusammengestellt.

Mehrfach wurde in der Arbeit darauf hingewiesen, daß ein Modellierungswerkzeug für eine schnelle Realisierung auf die Verwendung kommerzieller Komponenten angewiesen ist. Auch die im Rahmen dieser Arbeit entstandene Modellierung und die im Mikrosystem und auf dem Hostrechner ablaufende Software wäre ohne die Verwendung von bereits bestehenden Programmpaketen und -bibliotheken nicht realisierbar gewesen; sie sollen daher in einer Zusammenstellung genannt werden. Die Liste umfaßt nicht Basissoftware wie Betriebssysteme oder gewöhnliche Hilfsprogramme wie Compiler, Editoren oder Debugger. Programme, die frei verfügbar sind oder die für Forschungszwecke unentgeltlich eingesetzt werden dürfen, sind durch jeweils durch ein (*) gekennzeichnet.

Abschließend wird die Funktionweise der im Mikrosystem ablaufenden Software und die Kommandos, mit der sie gesteuert werden kann, erklärt.

A Glossar

µP, µC	Mikroprozessor bzw. Mikrocontroller.
ADC	Analog-Digital-Converter, ein elektronischer Baustein, welcher analoge Spannungen in binäre Zahlenwerten konvertieren kann. Die Auflösung wird durch die Zahl der dabei verwendeten Bits festgelegt.
Aktor	Gerät, welches eine elektrische Größe in eine andere physikalische Größe umwandelt.
Browser	Werkzeug, mit dem eine größere Datenmenge so dargestellt werden kann, daß das Auffinden einer gewünschten Information erleichtert wird.
C	weitverbreitete prozedurale Programmiersprache.
C++	eine um objektorientierte Elemente erweiterte Variante von C, die am weitesten verbreitete objektorientierte Sprache.
ChemFET	Chemisch sensitiver Feldeffekt-Transistor
EPROM	Erasable Programmable ROM (siehe dort).
ER-Modell	Entity-Relationship-Modell, ein semantisches Datenmodell, welches oft beim Entwurf von relationalen Datenbanken eingesetzt wird.
Expertensystemshell	Begriff für eine Klasse von Werkzeugen aus dem Bereich der KI, welche verschiedene Verfahren zur Wissensrepräsentation bereitstellen, um menschliches Wissen so zu formulieren, daß es geeignet weiterverarbeitet werden kann.
FEM	Finite-Elemente Methoden, ein Näherungsverfahren zur Berechnung kontinuierlicher physikalischer Systeme.
Frame	Struktur, der eine Menge von Eigenschaften in Form von Attributen und Werten zugeordnet ist.
Hypertext	eine Technik, bei dem Teile eines Textes nicht in linearer Folge zusammenhängen, sondern als Knoten eines Graphen angeordnet sind, die durch Links verbunden sind.
Instanziierung	Bildung eines Objektes als Teil einer Klasse unter Übernahme der Klasseneigenschaften.
Inferenz	Erschließen von neuem Wissen aus vorhandenem Wissen. Ist sehr stark von der verwendeten Wissensrepräsentationsform (Regeln, Logik, Frames etc.) abhängig.
Interrupt	Unterbrechung eines laufenden Programms durch ein externes oder internes Ereignis.
KfK	Kernforschungszentrum Karlsruhe.

KI	Künstliche Intelligenz, ein Teilgebiet der Informatik.
Komponentenmodell	bezeichnet in dieser Arbeit ein Verhaltensmodell einer Mikrosystemkomponente, das bei ausreichender Genauigkeit mit so wenig Aufwand berechnet werden kann, daß es im Rahmen eines Systementwurfs eingesetzt werden kann. Oft verbirgt sich dahinter die idealisierte Beschreibung eines physikalischen Vorgangs durch Differentialgleichungen, deren Lösung als Unterprogramme einer Programmiersprache formuliert werden können.
LAN	Local Area Network, dient zur Vernetzung von Rechnern in einem Bereich von wenigen km.
LIGA	Akronym für LIthografie, Galvanoformung und Abformung, ein Mikrostrukturierungsverfahren.
Mikrosystemtechnik	Technik, die sich mit dem Entwurf und der Fertigung von Mikrosystemen beschäftigt.
Modell	bezeichnet in dieser Arbeit die rechnerbasierte Abbildung von systemrelevanten Merkmalen von Mikrosystemen auf der Grundlage einer objektorientierten Modellierung.
Modellierung	Tätigkeit des Entwickelns eines Modells oder einer modellhaften Vorstellung.
MUX	Multiplexer, ein Baustein, welcher mehrere Eingänge zeitlich versetzt auf einen Ausgang schaltet.
OO	Objektorientiert, Objektorientierung.
OODBS	Objektorientiertes Datenbanksystem, speichert Daten auf der Basis eines objektorientierten Datenmodells.
Paradigma	die Menge aller Erfahrungen, Theorien und Axiome innerhalb eines Bereichs einer wissenschaftlichen Disziplin.
RAM	Random Acess Memory, schneller Speicher für Schreib- und Lesezugriffe.
ROM	Read Only Memory, schneller Speicher, der nur gelesen werden kann.
RS-232	verbreitete serielle Verbindung zur digitalen Übertragung von Daten.
SCCS	Source Code Control System, ein unter UNIX weit verbreitetes Werkzeug zur Versionsverwaltung von Textdateien, die es gestattet, ältere Versionen eines Dokuments wieder zu restaurieren.
Semantik	Bedeutung, Inhalt.
Sensor	Gerät, welches eine physikalische Größe in eine elektrische Größe umwandelt.

SMD	Surface Mounted Device, eine Technik zur Bestückung von Leiterplatten.
Timer	elektronischer Baustein, welcher Zeitmessungen durchführen kann.
UNIX®	verbreitetes Betriebssystem mit Multiuser- und Multitaskingeigenschaften.

B Liste der verwendeten Software

(die mit (*) gekennzeichneten Programme sind für Forschungszwecke frei verfügbar)

LEDA(*) ist eine C++-Klassenbibliothek, die häufig verwendete Datentypen wie Felder, Listen und Bäume zur Verfügung stellt. LEDA wurde am Fachbereich Informatik der Universität des Saarlandes von Kurt Mehlhorn und Stefan Näher entwickelt.

PLPLOT(*) ist eine Bibliothek mit Routinen zur grafischen Darstellung von Meßdaten in x,y-Koordinatensystemen. Die verwendete Version 4.0 wurde hauptsächlich von Maurice LeBrun und Geoff Furnish an der University of Texas entwickelt.

X11(*) oder das *X Window System, Version 11* ist eine Software zur Implementierung von grafischen netzwerktransparenten Oberflächen nach dem Client-Server-Prinzip. Sie wurde am MIT (Massachusetts Institute of Technology) entwickelt und ist Grundlage der Implementierung von Benutzeroberflächen auf den Workstations aller namhaften Hersteller.

XView(*) ist ein Toolkit von Sun Microsystems, Inc., welches die Erstellung von grafischen Benutzeroberflächen unter X11 erleichtert, indem es vordefinierte Elemente wie Buttons, Menus und Scrollbars zur Verfügung stellt.

SlingShot(*) ist eine Erweiterung von XView und wurde von Brian Warkentine bei SunSoft, Inc. entwickelt.

xloadimage(*) von Jim Frost ist ein Programm, um Bilder, die in verschiedenen Formaten gespeichert sind, unter X11 anzuzeigen.

OpenWindows Developer's Guide wurde von Sun Microsystems, Inc. entwickelt. Es ist ein grafischer Editor zum Erstellen von Benutzeroberflächen.

80C166 Compiler Package und *Cross View HLL-Debugger für 80C166* der Firma TASKING SOFTWARE DEUTSCHLAND GmbH gestatten die Programmentwicklung für den SAB 80C166 Mikrocontroller auf einer Sun-Workstation und ein Austesten von Software im Zielsystem unter Verwendung eines Monitorprogramms.

Das *C166 Professional Developers-Kit* der Firma KEIL ELEKTRONIK GmbH umfaßt Compiler, Assembler, Linker, Locator und einen Simulator zum Austesten von Programmen. Es gestattet eine Softwareentwicklung für den SAB 80C166 Mikrocontroller auf einem PC.

Nexpert Object ist eine Expertensystemshell der Firma Neuron Data, Inc. Eine Beschreibung findet sich in Kapitel 3 der Arbeit.

C Beschreibung des im Mikrosystem eingesetzten Kernels[1]

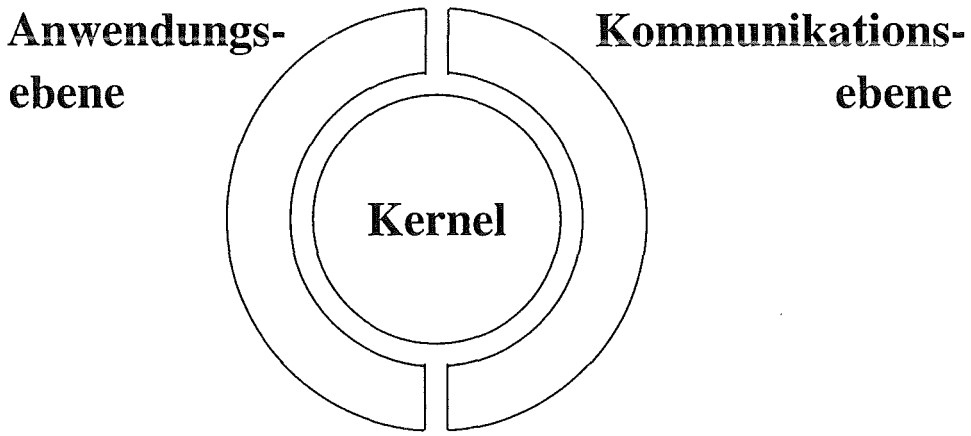
Für die Sensorsignalverarbeitung und Steuerung der Systemkomponenten des in Kapitel 4 beschriebenen Mikrosystems wurde ein Kernel entwickelt, welcher wesentliche Systemfunktionen interruptgesteuert zur Verfügung stellt. Die Funktion dieses Kernels ähnelt der eines Speicheroszilloskops.

Der Kernel wurde auf einem Mikrocontroller (SAB 80C166) implementiert. Seine wichtigsten Merkmale sind:

- 20 MHz maximaler Prozessortakt
- 256 KByte Adreßraum
- 5 16-bit-Zähler
- 2 serielle Schnittstellen
- 10-Bit Analog/Digital-Wandler (Wandlungszeit 10 µs) mit 10 Analogeingängen und Eingangsmultiplexer
- ein 10-bit Port (Input)
- vier 16-bit Ports (Input/Output)
- Interruptcontroller mit 16 Prioritätsebenen und 19 externen Interrupteingängen
- Peripheral Event Controller
- 4-stufige Befehls-Pipeline (parallele Abarbeitung von bis zu vier Befehlen)

Die Anbindung übergeordneter Systeme geschieht über zwei serielle Schnittstellen, die im Mikrocontroller integriert sind. Über eine Schnittstelle findet der normale Datenaustausch statt, während die zweite für andere Zwecke, etwa Debugging, Diagnose oder Tests verwendet werden kann.

Über das Kernprogramm können eine Anwendungsebene und eine Kommunikationsebene gesetzt werden (siehe Abbildung). Beide sind in ihrer Realisierung anwendungsspezifisch. Die



Kommunikationsebene stellt den Kontakt zur Außenwelt über eine RS232-Schnittstelle bereit. Über die Kommunikationsebene werden Programme der Anwendungsebene initialisiert. Anwendungsprogramme können die Routinen des Kernels benutzen.

1. Anhang C und D sind nicht Bestandteil der an der Universität eingereichten Dissertation

Der Kernel ist in "C" erstellt und kann vom Konzept her auf andere Mikrocontroller übertragen werden. Ein CPU-unabhängiger Datentransfer Controller als Bestandteil eines Mikrocontrollers ist dabei wie im vorliegenden Falle des 80C166 vorteilhaft, aber nicht notwendig.

Der Kernel ist nur bei Bedarf aktiv. Seine Aktivität wird über Interrupts angefordert. Die Komponenten des Mikrocontrollers, die eine Aktivität des Kernels anfordern können, sind die serielle Einheit, der Analog/Digitalwandler (ADC) über den Datentransfercontroller (PEC) sowie diverse Timer.

Ein Timer des Controllers generiert die Zeitbasis für die Abtastrate. Ein Interrupt dieser Einheit veranlaßt die CPU, den ADC zu starten.

Konvertierte ADC-Werte werden zunächst ohne CPU-Aktivität in einen Zwischenspeicher transferiert. Sind so die Meßwerte aller gewünschten Sensoren registriert, übernimmt die CPU des Mikrocontrollers angeregt durch einen Interrupt die weitere Bearbeitung und Speicherung der Daten.

Ein weiterer Timer fungiert als interne Systemuhr. Eine damit korrelierte, interruptfähige Einheit kann den Beginn der Datenaufzeichnung zu einer bestimmten Zeit auslösen. Ebenfalls in Kombination mit diesem Timer kann ein externes Triggerereignis die CPU aktivieren, der Timer hält hierbei den Triggerzeitpunkt fest.

Schließlich erfordert die serielle Einheit der Terminalschnittstelle des Kernels die Aktivität des Datentransferkontrollers PEC sowie der CPU selbst.

Im Rahmen der Software-Entwicklung wurde im wesentlichen die Methode der Software-Simulation des gesamten Controllers auf einem PC angewandt. Da hierzu - im Gegensatz zur Emulation des Controllers - kein Zielsystem notwendig ist, konnte die Entwicklung des Programmpakets parallel zur Hardware-Entwicklung stattfinden.

C.1 Beschreibung einzelner Funktionen

Der Kernel führt folgende hardwarenahe Funktionen im Mikrosystem aus:

- Aufnahme der Sensordaten in verschiedene Modi, einfache Vorverarbeitung der Daten, Korrektur von Nullpunktabweichungen der Sensoren, primitive Selbsttestfunktion für die Sensoren,
- Systemzeit mit langer Laufdauer und hoher Auflösung,
- RS232-Schnittstelle zur Systemwartung mit einfacher Kommandosprache.

Datenaufnahme

Das Konzept der Datenaufnahme ähnelt dem eines digitalen Speicheroszilloskopes. Für die Aufzeichnung der Daten in einem Pufferspeicher sind mehrere Modi vorgesehen:

- Aufzeichnung ohne Vorbedingung (trigger free running)
- Aufzeichnung nach Erreichen eines Schwellwertes (trigger level)

- Aufzeichnung ab einem bestimmten Zeitpunkt (trigger time)
- Aufzeichnung wird durch ein externes Ereignis gestartet (trigger extern)

Jeder dieser Aufnahmemodi kann einmalig (single shot mode) oder wiederholt (continuous mode) erfolgen. Dabei können die Daten nur eines einzelnen Sensors (single channel mode), aller sechs Sensoren (multi channel mode) oder der Mittelwert von je drei Sensoren registriert werden (average channel mode). Weiterhin kann die Triggervorgeschichte aufgezeichnet werden (trigger history) und der Trigger verzögert werden (trigger delay).

Zur Datenspeicherung stehen je nach gewählter Speichertiefe ein oder zwei getrennte Puffer zur Verfügung. Bei zwei Datenspeichern besteht so die Möglichkeit, die Daten eines Puffers zu bearbeiten (übergeordnetes Anwendungsprogramm, Übertragung der Daten über die serielle Verbindung) und parallel dazu erneut Daten im zweiten Puffer aufzuzeichnen.

Systemzeit

Ein Timer des Mikrocontrollers bildet eine Echtzeit-Uhr mit hoher Auflösung und langer Laufdauer (etwa 41,4 Monate bei 400ns Auflösung). Damit lassen sich sowohl Kurzzeitereignisse als auch Langzeitmessung mit einer hohen zeitlichen Präzision erfassen.

Schnittstellen

Unabhängig von einer übergeordneten Kommunikationsebene verfügt der Kernel über eine eigene "Monitor"-Schnittstelle (RS232C, Vollduplex, XON/XOFF, 9600 baud). An diese Schnittstelle kann ein Terminal oder ein Leitrechner angeschlossen werden. Mit einer einfachen Kommandosprache besteht Zugriff auf alle Funktionen des Kernels.

Zu jedem Befehl paßt eine zugehörige C-Funktion, die von einem Anwendungsprogramm aus aufgerufen werden kann und der gleichen Syntax unterliegt wie das entsprechende Kommando der Terminalschnittstelle.

D Beschreibung der Kommandos des Kernels

Die Kommandos, mit denen der Kernel gesteuert wird, lassen sich nach ihrer Funktionalität in 4 Gruppen einteilen:

- Steuerbefehle
- Systemparameter definieren
- Datenanforderung
- Modus der Datenaufnahme definieren

Diese Kommandos können sowohl von einer Anwendungsebene als auch direkt durch einen Anwender über die serielle Schnittstelle aufgerufen werden.

D.1 Steuerbefehle

Kommando: **PON** **Power on**

Parameter: *keine*

Funktion: startet eine Datenaufzeichnung. Zur Vorbereitung muß der gewünschte Modus eingestellt werden (siehe TEX, TFR, TLV,TTM). Die Datenaufzeichnung wird entweder nach einer einmaligen Messung beendet (siehe TSM) oder automatisch immer wieder neu gestartet (siehe TCM). In diesem Fall wird PON zum Beenden der Datenaufzeichnung verwendet.

Kommando: **POF** **Power Off**

Parameter: *keine*

Funktion: stoppt eine laufende Datenaufzeichnung.

Kommando: **EON** **Echo On**

Parameter: *keine*

Funktion: bewirkt, daß jedes über die serielle Schnittstelle empfangene Zeichen zur Bestätigung wieder zurückgeschickt wird.

Kommando: **EOF** **Echo Off**

Parameter: *keine*

Funktion: bewirkt, daß über die serielle Schnittstelle empfangene Zeichen nicht zurückgeschickt werden.

Kommando: **SRE** **System Reset**

Parameter: *keine*

Funktion: bewirkt einen Reset des gesamten Systems. Gespeicherte Daten gehen verloren, eine laufende Datenaufzeichnung wird gestoppt. Die Systemzeit wird auf 0 zurückgesetzt. Einstellungen des Datenaufzeichnungsmodus werden auf ihre Defaultwerte zurückgesetzt.

Kommando: **SON** **Selftesting On**

Parameter: *keine*

Funktion: bewirkt, daß die Meßwerte von jeweils in einer Raumrichtung messenden Sensoren untereinander auf Übereinstimmung geprüft werden. Zu stark von anderen Sensoren abweichende Messungen führen dazu, daß ein Sensor als

fehlerhaft markiert wird. Ein solcher Sensor wird nicht mehr bei der Triggerlevelüberwachung eingesetzt und im Average Channel Modus nicht für die Berechnung des Mittelwerte verwendet.

Kommando:	**SOF**	**Selftesting Off**
Parameter:	*keine*	
Funktion:	die Überprüfung von Sensoren untereinander wird ausgeschaltet, es findet kein Selbsttest mehr statt.	

Kommando:	**MON**	**Sensor Masking On**
Parameter:	*unsigned int*	*Nummer (0-5) eines Sensors*
Funktion:	ein Sensor wird als fehlerhaft markiert. Markierte Sensoren werden nicht zur Triggerlevelüberwachung herangezogen, und im Average Channel Mode nicht zur Berechnung des Mittelwertes verwendet. Im SCM oder MCM Datenaufzeichnungsmodus kann aber trotzdem auf die von ihnen stammmenden Meßergebnisse zugegriffen werden.	

Kommando:	**MOF**	**Sensor Masking Off**
Parameter:	*unsigned int*	*Nummer (0-5) eines Sensors*
Funktion:	hebt die Maskierung eines Sensors auf, der als fehlerhaft markiert war. Der Sensor wird wieder zur Triggerlevelüberwachung eingesetzt, und im Average Channel Mode zur Mittelwertbildung verwendet.	

Kommando:	**QER s**	**Quit Error Flag**
Parameter:	*unsigned int*	*Nummer (0-5) eines Sensors*
Funktion:	hebt die Maskierung eines Sensors auf, der durch die Selbstüberwachung als fehlerhaft erkannt worden war. Der Sensor wird wieder zur Triggerlevelüberwachung eingesetzt, und im Average Channel Mode zur Mittelwertbildung verwendet.	

Kommando:	**SCM**	**Single Channel Mode**
Parameter:	*unsigned int*	*Nummer (0-5) eines Sensors*
Funktion:	es werden nur Daten eines Sensors aufgezeichnet. Ist sinnvoll bei einer Datenaufzeichnung mit einer sehr hohen Geschwindigkeit oder einem sehr hohen Speicherbedarf oder zur gezielten Überprüfung eines einzelnen Sensors.	

Kommando:	**ACM**	**Average Channel Mode**
Parameter:	*keine*	
Funktion:	es werden alle Sensoren, die nicht als fehlerhaft markiert sind, für die Messung verwendet, aber nur der Mittelwert der in einer Raumrichtung messenden Sensoren wird aufgezeichnet.	

Kommando:	**MCM**	**Multiple Channel Mode**
Parameter:	*keine*	
Funktion:	für die Messung werden alle Sensoren verwendet und die Daten der einzelnen Sensoren werden aufgezeichnet. Sinnvoll, um das System zu testen, erfordert den meisten Speicherplatz aller Datenaufzeichnungsmodi.	

D.2 Systemparameter definieren

Kommando:	**CTM**	**Calibrate Time**
Parameter:	*unsigned long*	*Systemzeit in 1/40 sec*
Funktion:	die Systemzeit wird auf den angegebenen Wert gesetzt. Kann verwendet werden, um die Systemzeit mit einer externen Uhr zu synchronisieren.	

Kommando:	**COF**	**Calibrate Offset**
Parameter:	*keine*	
Funktion:	die Daten der letzten Messung werden dazu verwendet, einen Offset der einzelnen Sensoren zu korrigieren. Damit dieses Kommando korrekt ausgeführt werden kann, muß die letzte Messung im Multiple Channel Mode ausgeführt worden sein, damit für alle Sensoren auch tatsächlich Meßdaten vorliegen.	

Kommando:	**CSO**	**Change Sensor Offset**
Parameter:	*unsigned int* *int*	*Nummer (0-5) eines Sensors* *Offsetwert des Sensors (-512 - 511)*
Funktion:	ändert den Offsetwert des Sensors auf den angegebenen Wert. Kann dazu verwendet werden, bekannte Offsets der Sensoren schon beim Systemstart zu korrigieren.	

Kommando:	**CUL**	**Calibrate Upper Level**
Parameter:	*unsigned int*	*Schwellwert (0 - 1023)*
Funktion:	setzt einen oberen Schwellwert für gespeicherte Meßwerte. Bei Überschreiten dieser Schwelle wird der betreffene Sensor als fehlerhaft markiert, falls die Selbstüberwachung aktiviert ist.	

Kommando:	**CLL**	**Calibrate Lower Level**
Parameter:	*unsigned int*	*Schwellwert (0 - 1023)*
Funktion:	setzt einen unteren Schwellwert für gespeicherte Meßwerte. Bei Unterschreiten dieser Schwelle wird der betreffene Sensor als fehlerhaft markiert, falls die Selbstüberwachung aktiviert ist.	

Kommando:	**CCS**	**Change Channel Size**
Parameter:	*unsigned int*	*Größe des Datenspeichers*
Funktion:	setzt die Größe des für eine Messung zur Verfügung stehenden Datenspeichers fest. Falls ein zu großer Wert angegeben wird, wird der maximal mögliche Wert gesetzt. Der zur Verfügung stehende Speicher kann mit RCS erfragt werden.	

D.3 Datenanforderung

Kommando:	**RSC**	**Request Single Channel**
Parameter:	*unsigned int*	*Nummer (0-5) eines Sensors*
Funktion:	die gemessenen Daten eines einzelnen Sensors werden über die serielle Schnittstelle ausgegeben. Falls für diesen Sensor keine Messdaten vorhanden sind, werden zwei Newline ausgegeben. Falls eine aktuelle Messung läuft, werden zuerst die bis dahin angefallenden Daten und dann die noch zu messenden Daten ausgegeben, sobald sie vorliegen. Die Übertragung kann durch eine weitere Datenanforderung jederzeit unterbrochen werden.	

Kommando:	**RMC**	**Request Multiple Channel**
Parameter:	*keine*	
Funktion:	gleiche Funktion wie RSC, nur werden die Daten aller Sensoren über die serielle Schnittstelle ausgegeben, falls vorhanden. Ansonsten werden zwei Newline ausgegeben. Daher kann dieser Befehl nur sinnvoll eingesetzt werden, wenn Daten im Multiple Channel Modus aufgezeichnet wurden.	

Kommando:	**RTM**	**Request Time**
Parameter:	*keine*	
Funktion:	die aktuelle Systemzeit wird ausgegeben. Das Format entspricht den Eingabeparametern von CTM.	

Kommando:	**ROF**	**Request Offset Data**
Parameter:	*keine*	
Funktion:	die aktuellen Werte zur Korrektur von Sensoroffsets aller Sensoren werden ausgegeben.	

Kommando:	**RTB**	**Request Time Base**
Parameter:	*keine*	
Funktion:	die für eine Messung verwendete Abtastrate der Sensoren wird ausgegeben. Das Format entspricht den Eingabeparametern von TBA.	

Kommando:	**RTE**	**Request Trigger Event**
Parameter:	*keine*	
Funktion:	es werden die Messwerte ausgegeben, die bei der Auslösung eines Triggers gemessen wurden.	

Kommando: **RSS** **Request System Status**

Parameter: *keine*

Funktion: es wird der aktuelle Systemstatus ausgegeben.

Kommando: **RCT** **Request Conversion Time**

Parameter: *keine*

Funktion: es wird die Zeit ausgegeben, bei der mit der Aufzeichnung des aktuellsten Datensatzes begonnen wurde. Das Format entspricht den Eingabeparametern von CTM.

Kommando: **RMS** **Request Memory Size**

Parameter: *keine*

Funktion: es wird die maximale für Meßdaten zur Verfügung stehende Speichergröße ausgegeben.

Kommando: **RCS** **Request Channel Size**

Parameter: *keine*

Funktion: es wird die aktuell für eine Messung zur Verfügung stehende Speichergröße ausgegeben.

Kommando: **RDA** **Request Data Acknowledge**

Parameter: *keine*

Funktion: hiermit quittiert der Benutzer dem Empfang von Meßdaten. Der durch diese Daten bisher belegte Speicher wird für eine neue Messung freigemacht.

D.4 Modus der Datenaufnahme definieren

Kommando:	**TBA**	**Time Base**
Parameter:	*unsigned int*	*Prescale(0-7)*
	unsigned int	*Basis (0 - 65535, in 200 nsec)*
Funktion:	gibt an, in welchem zeitlichen Abstand Daten während einer Messung aufgezeichnet werden. Die durch die Basis vorgegebene Zeit kann durch Setzen des Precales verlängert werden: Zeit = Basis * $2^{Prescale}$. Die minimal mögliche Zeitbasis hängt davon ab, von wievielen Sensoren Meßwerte aufgenommen werden sollen.	

Kommando:	**TSM**	**Trigger Single Shot Mode**
Parameter:	*keine*	
Funktion:	der aktuell für Messdaten zur Verfügung stehende Speicher wird nach dem Beginn der Messung aufgefüllt, danach finden keine weiteren Messungen mehr statt.	

Kommando:	**TCM**	**Trigger Continuous Mode**
Parameter:	*keine*	
Funktion:	der aktuell für Messdaten zur Verfügung stehende Speicher wird nach dem Beginn der Messung aufgefüllt, danach wird auf den nächsten freien Speicher umgeschaltet und eine erneute Messung gestartet.	

Kommando:	**TDL**	**Trigger Delay**
Parameter:	*unsigned long*	*Zeit in 1/40 sec*
	unsigned int	*Zeit (0 - 50000, in 400 nsec)*
Funktion:	nach dem Eintreffen eines Triggerereignisses wird noch die angegebene Zeitspanne gewartet, bis mit einer Messung begonnen wird.	

Kommando:	**TPO**	**Trigger Position**
Parameter:	*unsigned int*	*Triggerposition*
Funktion:	falls die Triggerposition von 0 verschieden ist, gibt sie an, wieviele Meßwerte vor dem Eintreffen eines Triggerereignisses durch das System gespeichert werden sollen. Dadurch kann eine Triggervorgeschichte aufgenommen werden. Eine eventuell mit TDL eingeschaltete Triggerverzögerung wird abgeschaltet.	

Kommando:	**TEX**	**Trigger Extern**
Parameter:	*keine*	
Funktion:	ermöglicht die Reaktion auf ein externes Triggerereignis, nach dessen Eintreten mit der Datenaufzeichnung begonnen wird.	

Kommando:	**TFR**	**Trigger Free Running**
Parameter:	*keine*	
Funktion:	schaltet eine freilaufende Datenaufzeichnung ohne Überwachung von Triggerereignissen ein.	

Kommando:	**TTM**	**Trigger Time**
Parameter:	*unsigned long*	*Systemzeit in 1/40 sec*
	unsigned int	*Systemzeit (0 - 50000) in 400 nsec*
Funktion:	setzt einen Zeitpunkt, zu dem mit einer Datenaufzeichnung begonnen werden soll.	

Kommando:	**TLV**	**Trigger Level**
Parameter:	*unsigned int*	*Schwellwert (0 - 1023)*
Funktion:	setzt einen Schwellwert, ab dessen Überschreitung Daten aufgezeichnet werden.	

Kommando:	**TRG**	**Trigger**
Parameter:	*keine*	
Funktion:	simuliert einen externen Trigger. Falls das System gerade im TEX-Modus arbeitet, wird mit der Datenaufzeichnung begonnen, ansonsten wird das Kommando ignoriert.	

Danksagung

Ich danke Herrn Prof. Dr. Menz vom Institut für Mikrostrukturtechnik des Forschungszentrums Karlsruhe für seine Bereitschaft, diese Arbeit als Doktorvater zu betreuen und als Hauptreferent zu vertreten.

Ebenfalls danke ich Herrn Prof. Dr. Trauboth für die Aufnahme am Institut für Angewandte Informatik des Forschungszentrums Karlsruhe und die Übernahme des Korreferats dieser Arbeit.

Besonderer Dank gebührt Herrn Dr. Eggert, welcher als Leiter der Abteilung Mikrosystem-Informatik die thematische Ausrichtung der Arbeit maßgeblich mitbestimmt und die fachliche Betreuung übernommen hat.

Herrn Dr. Süß schulde ich Dank für seine Unterstützung bei der Entwicklung des Kernels für das Mikrosystem.

Ich möchte allen Personen danken, die an der Entwicklung des Mikrosystems zur Messung von Beschleunigungen beteiligt waren. Insbesondere danke ich Herrn Dr. Strohrmann (Institut für Mikrostrukturtechnik) für die Beantwortung vieler Fragen zum LIGA-Beschleunigungssensor, und Herrn Kühner (Hauptabteilung Prozeßdatenverarbeitung und Elektronik) für die Entwicklung des Mikrocontrollermoduls.

Herrn Dr. Düpmeier (Institut für Angewandte Informatik) danke ich für die engagierte und kompetente Betreuung der Abteilungsrechner.