Modelle und Methoden für den integrierten Systementwurf

Franz J. Rammig, Universität-GH Paderborn, Cadlab

1 Kurzfassung

In diesem Beitrag soll versucht werden, Ansätze des integrierten Entwurfs heterogener Gesamtsysteme zu beschreiben. Während in der Vergangenheit die verschiedenen Ingenieursdisziplinen relativ isoliert gearbeitet haben, besteht heute die Notwendigkeit, ein komplettes System in der Gesamtheit seiner Komponenten (Mechanik, Elektronik, Software) integriert zu betrachten und zu entwerfen. Dies stellt zunächst neue Anforderungen bezüglich der Spezifikation und der Modellierung derartiger Gesamtsysteme. Im ersten Teil des Beitrages werden daher verschiedene Spezifikations- und Modellierungsansätze vorgestellt, die geeignet sein können, Gesamtsysteme zu behandeln. Steht ein geeignetes Spezifikations- und Modellierungsgerüst zur Verfügung, kann eine Methodik des Systementwurfs erarbeitet werden. Darauf basierend können dann Werkzeuge entwickelt werden. Diesem Themenkomplex ist der zweite Teil des Beitrages gewidmet. Dabei ist es wesentlich, keine völlig neue Werkzeugwelt entstehen zu lassen, sondern bewährte Vorgehensweisen in den verschiedenen Ingenieursdisziplinen geeignet zu kombinieren. Dies führt zum Ansatz des "Concurrent Engineering". Darauf wird ebenfalls eingegangen.

2 Ziele und Aufgaben des integrierten Systementwurfs

Der Entwurf digitaler Hardware über verschiedene Entwurfsebenen ist heute sehr gut verstanden und durch Werkzeuge unterstützt. Dabei war sicherlich sehr hilfreich, daß in diesem Bereich relativ früh ein Gerüst wohldefinierter Abstraktionsebenen und Sichten entstanden ist. Somit konnte man die Entwurfsaufgabe in überschaubare und klar umreißbare Teilaktivitäten partitionieren. Für diese Teilaufgaben wurden schrittweise Methoden und Werkzeuge erarbeitet, wobei die Entwicklungsrichtung "bottom up" war, d.h. die Werkzeugunterstützung begann auf niedrigem Abstraktionsniveau, und bezüglich der Sichten (nach Gajski /11/ Verhalten, Struktur und Geometrie) lag der Schwerpunkt zu Beginn eher bei den Sichten Struktur und Geometrie, um sich dann schrittweise mehr der Verhaltenssicht zuzuwenden. Welches sind nun die verschiedenen Abstraktionsebenen? Sie sollen hier aus der Verhaltenssicht heraus kurz charakterisiert werden, d.h. eine Abstraktionsebene ergibt sich aus einem für diese Ebene spezifischen Modellierungskonzept.

Auf der untersten Ebene, der elektrischen Ebene, wird mittels Differentialgleichungen modelliert, wie sich das System elektrisch über die Zeitachse verhält, wobei man für die Zeitachse wie auch für die beobachtbaren Werte gleichermaßen kontinuierliche

Wertebereiche annimmt. Aus Sicht der Geometrie handelt es sich um die Layoutebene.

Abstrahiert man elektrische Komponenten zu idealen Schaltern und idealisierten Kapazitäten, so hat man die Schalterebene erhalten. Hier modelliert man die Zeit weiterhin kontinuierlich, die beobachtbaren Werte jedoch werden durch diskretwertige Paare (logischer Wert, Stärke) dargestellt. Die Entsprechung in der Geometriesicht sind Stickdiagramme.

Beschränkt man sich unter allen möglichen Transistorschaltungen auf solche, die eine Entsprechung in der Aussagenlogik haben (logische Gatter), so hat man die Gatterebene erreicht. Zwar arbeitet man hier meist immer noch mit kontinuierlicher Zeit, doch der Wertebereich der beobachtbaren Werte ist definitv diskret, im Idealfall zweiwertig. Als Modelliermethode hat man hier ein System Boolescher Gleichungen vorliegen.

Der nächste Abstraktionschritt ist dahingehend interessant, daß nicht nur eine weitere Abstraktion der Elementarbausteine stattfindet (Register statt einzelne Flipflops, ALUs statt einzelner Gatter) sodern auch eine bestimmte Betriebsweise unterstellt wird (Registertransfers), sodaß nur noch die Wirkung dieser Transfers zu spezifizieren ist. Konsequenterweise heißt diese Ebene auch Registertransferebene. Hier wird nun auch die Zeit diskret modelliert (Zählen von Takten).

Auch der nächste Abstraktionsschritt ist weitgehend von einer Verhaltensabstraktion geprägt. wird auf der RT-Ebene das System aus der Sicht der Einzelkomponenten reaktiv gesehen, so nimmt die algorithmische Ebene die imperative Sicht der Steuerung ein. Die für die Abarbeitung eines (hochgradig nebenläufigen) Algorithmus notwendigen Mechanismen werden hier als gegeben angesehen, sodaß man sich auf die Formulierung dieses Algorithmus beschränken kann. Auf dieser Ebene verläßt man häufig auch die Betrachtung realer Zeit und zieht sich auf Kausalzusammenhänge zurück.

Auf der Systemebene schließlich abstrahiert man auch von der algorithmischen Realisierung der Systemfunktionen. Hier benennt man nur die Akteure des Gesamtsystems, die von ihnen angebotenen Dienste (Methoden im Sinne der objektorientierten Programmierung) und das Geflecht der wechselseitigen Anforderung und Erbringung dieser Dienste. Wie schon auf der algorithmischen Ebene arbeitet man hier fast ausschließlich mit einer logischen Zeit, die sich allein durch Kausalzusammenhänge ergibt.

Den Übergängen zwischen den Abstraktionsebenen sind ("top down") Methoden und Werkzeuge zugeordnet. Der (derzeit durch Werkzeuge wenig unterstützte) Systementwurf bildet die objektorientierte, funktionale Sicht in Algorithmen ab. Der Übergang von der algorithmischen Ebene auf die RT-Ebene wird üblicherweise High-Level-Synthese (HLS) genannt. Entsprechende Werkzeuge diffundieren derzeit zunehmend aus dem akademischen Bereich in die industrielle Praxis. Mit der Logiksynthese wird der Übergang von der RT-Ebene auf die Gatterebene realisiert. Hier existieren seit längerer Zeit ausgereifte kommerzielle Werkzeuge. Die Umsetzung von Netzlisten auf der Logikebene in physikalische Realisierungen in einer bestimmten Technologie wird vom physikalischen Entwurf geleistet. Hier besonders kann auf ausgereifte Werkzeuge mit langer Tradition zurückgegriffen werden.

So ausgereift der Entwurf digitaler Hardware ist, so isoliert ist er auch. Dies mag für den Entwurf von Prozessoren tolerabel sein. Hier gilt es letztlich, eine Programmierschnittstelle, d.h. einen Interpretationsalgorithmus für einen Instruktionssatz be-

reitzustellen. Ansonsten ist der Prozessor, trotz seiner Einbindung in die Peripherie (die er dominiert) ein recht isoliertes Objekt. Nur spielt in Europa die Entwicklung von Prozessoren eine geringe Rolle. Digitale Hardware ist bei uns in aller Regel als eingebettetes System zur Steuerung oder Regelung eines umfassenderen Systems zu sehen. Hierbei spielt die auf der Hardware laufende Software eine sehr gewichtige Rolle. Eine in toto optimale Lösung wird sich also nur finden lassen, wenn man von vornherein das System als Gesamtsystem plant, aus der Spezifikation des Gesamtsystems herausarbeitet, was durch eine spezielle Elektronik mit darauf laufender Software realisiert werden soll und diesen Teil dann weiterhin zwischen Hardware und Software balanciert. Der Antriebsstrang eines Automobils mag als Beispiel dienen. Selbstverständlich kann man den gesamten Antriebsstrang einschließlich der Regelung in reiner Mechanik realisieren. Man kann aber auch das Motormanagement einer Elektronik übertragen, die ihrerseits die gesamte Bandbreite vor reiner Hardwarelösung bis hin zu frei programmierbaren Standardprozessoren, wo dann der Algorithmus als Programm in einem ROM steht, überdecken kann. Man kann natürlich noch weiter gehen und den mechanischen Strang, der Energie und Information überträgt, entkoppeln und durch einen Energiestrang, der ein elektrisches Energieangebot unterbreitet, und einen Informationsstrang, der die notwendige Regelungsinformation zu den an den Rädern angebrachten Elektromotoren transportiert, ersetzen. Dieses Beispiel zeigt auch bereits, daß man bei der Entwicklung eingebetteter Elektronik sinnvollerweise frühzeitig an parallele Systeme denken muß. In zunehmenden Maße werden technische Systeme von gekoppelten Multiprozessorsystemen gesteuert und geregelt, d.h. die klassische Konfiguration einer zentralen Elektronik die über Feldbusse verschiedene Komponenten steuert, wird ersetzt durch eine Konfiguration, bei der die Intelligenz vor Ort angesiedelt ist. Die lokalen, intelligenten Steuerungskomponenten tauschen dann über schmalbandige Kanäle nur noch Kooperationsinformation aus. Man erkennt sofort die Analogie zur Modellvorstellung der Systemebene, wie wir sie oben für den (isolierten) Elektronikentwurf eingeführt haben.

Das Beispiel mag erläutert haben, wie verschiedene, bisher klassischerweise isoliert betrachtete Ingenieursdisziplinen kooperativ zusammenwirken müssen, um ein Gesamtsystem funktions- und kostenoptimal realisieren zu können. Der Mechaniker muß hinterfragen, wo er Energie- und Informationsströme entkoppeln kann und sollte (z.B. "fly by wire" anstatt mechanischer Kopplung zwischen Ruderhorn und Ruderklappe), wo er mechanische Komponenten entfeinern kann und sollte. Der Informationstechniker muß die dabei entstehenden Informationsströme analysieren, die darauf auszuführenden Algorithmen herauskristallisieren und plazieren. Schließlich muß er die lokalen Algorithmen in einer Kombination von Hardware und Software realisieren und die Kommunikation zwischen den Verarbeitungsknoten installieren. Hierbei sind Themenfelder anzusprechen, die klassischerweise eher im Bereich der Betriebssysteme angesiedelt waren (Scheduling paralleler Prozesse auf parallelen Prozessoren, Realzeitkommunikation, Kommunikationsprotokolle).

Es sollte aber auch deutlich geworden sein, daß es wenig sinnvoll ist, eine völlig neue Ingenieursdisziplin "Integrierter Systementwurf" zu kreieren. Vielmehr ist eine kooperative Vorgehensweise klassischer Disziplinen nicht nur ausreichend, sondern sogar vorzuziehen. Was allerdings als Klammer unbedingt erforderlich ist, ist eine, möglicherweise multiparadigmatische, Gesamtmodellierungsgrundlage. Ohne eine derarti-

ge Grundlage ist es gar nicht möglich, Alternativen, die Gewichte in verschiedene Ingenieursdisziplinen verlagern, zu explorieren. Dabei ist es nicht zwingend, daß ein derartiges umfassendes Modellierungskonzept als Schnittstelle den einzelnen Ingenieuren angeboten wird. Vielmehr kann das umfassende Modellierungskonzept als semantische Basis mit abstrakter Syntax im Hintergrund (d.h. intern in Entwicklungsrechnern) gehalten werden. Es muß nur präzise definiert sein, wie die dedizierten Modellierungsschnittstellen der verschiedenen Ingenieursdisziplinen in das Gesamtmodellkonzept abgebildet werden kann. Jede Disziplin wird dabei natürlich nur Teilaspekte abdecken, Teilaspekte, die aber einen weiten Überlapp haben. Nur durch diesen Überlapp ist ein Informationsaustausch zwischen den verschiedenen Ingenieursdisziplinen und eine Realisierungsverlagerung von Konzepten möglich. Die Suche nach einem derartigen umfassenden Modellierungskonzept ist daher ein zentraler erster Schritt auf dem Weg zum integrierten Systementwurf. Hier ist naturgemäß die Informatik als Wissenschaft von Informationsdarstellung und -verarbeitung zentral gefragt. Gleichwohl dürfen Traditionen aus anderen Diszipinen nicht übersehen werden. So hat sich die Informatik beispielsweise relativ wenig mit der Modellierung kontinuierlicher Systeme befaßt.

Liegt der Modellierungsrahmen fest, so kann mit Entwurfsmethoden und darauf aufbauend mit Entwurfswerkzeugen gearbeitet werden. Ein Entwurfswerkzeug ist letztlich nichts anderes als eine Transformation eines Modells des Entwurfsobjektes in ein anderes. Im Falle des integrierten Systementwurfs bedeutet dies zunächst, im Gesamtmodell Teilaktivitäten so zu verlagern, daß sich die Projektionen, die verschiedenen Ingenieursdisziplinen weitgehend entsprechen, identifizieren lassen und dabei eine Lösung zu finden, die unter einer globalen Kostenfunktion "optimal" ist. Dies ist die Phase der Systemanalyse mit der resultierenden Systempartitionierung. Ist dieser zentrale Schritt des Systementwurfs geleistet, können die verschiedenen involvierten Disziplinen nach ihrer lokalen Methodik kooperativ arbeiten. Kooperativ bedeutet hier einerseits weitgehende Autonomie aber Wissen über Interdependenzen. So mag beim Mechanikentwurf aus thermischen Gründen entschieden werden, ein stark perforiertes Gehäuse zu benutzen. Eine derartige Entscheidung muß aber mit dem Elektronikentwurf abgeglichen werden, um zu vermeiden, daß wegen evtl. höherer Ein-/Abstrahlung EMV-Probleme auftauchen.

3 Spezifikation und Systemmodellierung

Spezifikation und Modellierung sind zwei eng verwandte Aktivitäten bei jeder Art von Entwurfsprozessen. Es kann sogar geschegen, daß identische Sprachmittel benutzt werden. Dennoch besteht ein wesentlicher semantischer Unterschied: Spezifikation beschreibt ein intendiertes System, Modellierung ein veorhandenes. Der Unterschied wird besonders deutlich, wenn man indeterministische Beschreibungen betrachtet. Nichtdeterministik bei einer Modellierung bedeutet, daß sich das beschriebene System tatsächlich indeterministisch verhält (zumindest auf der Abstraktionsebene der Beschreibung). Im Falle einer Spezifikation bedeutet Indeterminismus dagegen eine indeterminiertheit der Entwurfsintention, d.h. für den darauf aufbauenden Entwurfsprozeß zusätzliche Freiheitsgrade. Ein kleines Beispiel mag dies erläutern: Wird in einem Modell die Verzögerung als "5 bis 10 ns" angegeben, so bedeutet dies, daß das beschriebene Objekt indeterministisch eine Verzögerung aus diesem Bereich annimmt. Die selbe Angabe in einer Spezifikation bedeutet, daß jede Verzögerungszeit in diesem Bereich

als korrekt angesehen wird und es dem Entwurfsprozeß freigestellt ist, eine passende zu wählen.

3.1 Graphische Methoden

Da heute nahezu jeder Computerarbeitsplatz mit einer hochauflösenden Graphik versehen ist, ist es nicht überraschend, daß graphische Spezifikations- und Modellierungsmethoden zunehmend an Bedeutung gewinnen. Eine Reihe dieser graphischen Methoden sind relativ alt, oder beruhen auf altbekannten Darstellungsarten, wie z. B. auf Zustandsdiagrammen endlicher Automaten. Die Tendenz, zeichenkettenorientierte Darstellungen bei rechnergestützten Anwendungen zu verdrängen wird aber erst in jüngerer Zeit sichtbar.

3.1.1 RT/SA

Structured Analysis (SA) /9/ ist eine relativ alte Technik aus dem Bereich des Software Engineering. Für den hier diskutierten Anwendungsbereich des integrierten Systementwurfs technischer Systeme ist eine Erweiterung, genannt RT/SA /34/, von Interesse. In RT/SA wird ein System durch eine Menge von Dataflow Diagrams, einem Data Dictionary und einer Menge von Transformation Descriptions beschrieben. Ein Dataflow Diagram ist eine Darstellung in Form eines gerichteten Graphen. Es gibt drei Knotentypen in einem derartigen Diagramm: Kreise um Aktionen (Prozesse) darzustellen, Balkenpaare zur Darstellung von Datenspeichern und Rechtecke für Anschlußstellen. Der Datenfluß wird durch gerichtete Kanten dargestellt. In der Abb. 1 sind die verschiedenen Symbole zu sehen.
Kanten und Knoten von SA Dataflow Diagrams können mit Bezeichnern annotiert werden. Auf diese Bezeichner kann in anderen SA Dokumenten Bezug genommen werden. Hier ist zunächst das Data Dictionary zu nennen. In diesem Dokument wird die Terminologie des Projekts festgelegt und zu den referierten Bezeichnern in den Dataflow Diagrams werden weitere Details vermerkt. Der Aufbau des Data Dictionary ist nicht im Detail festgelegt; in der SA-Literatur werden reguläre Ausdrücke aber auch Prosatext angeben.

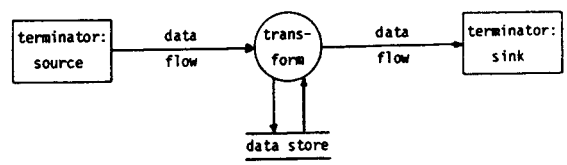

Abb.1: Graphische Darstellung von SA

Der nächste Typ zusätzlicher Dokumente ist der der Transformation Description, oft Minispec genannt. Ein Minispec beschreibt detailliert die internen Operationen einer Funktion innerhalb eines SA Dataflow Diagrams. Sie enthalten aber eben nur die interne Funktionalität, es wird keinerlei globale Information in Minispecs bereitgestellt. Auch hier ist der Formalismus offen, von Prosatext über Pseudocode bis hin zu formalen Detailspezifikationen. Als Beispiel in unserem Kontext wäre hier VHDL zu nennen, wie in /32/ vorgeschlagen. In RT/SA kommen zu den bisher beschriebenen datenflußorientierten Konzepten Controlflow Diagrams und Control Specifications hinzu. Dabei werden die beiden Sichten (Datenfluß und Kontrollfluß) über gleichnamige Knoten in den verschiedenen Diagrammen aneinander gebunden. Dies kann im Extremfall zu isolierten Knoten in einem der Graphen führen, z.B. wenn ein Kontrollflußknoten nicht in den Datenfluß eingebettet ist. Die syntaktische Form von Controlflow Diagrams ist identisch der der Dataflow Diagrams. An Stelle der Minispecs treten hier Control Specifications hinzu, um Details innerhalb von Steuerknoten anzugeben. Auch hier können verschiedene Formalismen benutzt werden, wobei auch für diesen Zweck unter anderem VHDL vorgeschlagen wurde

/32/. RT/SA-Spezifikationen können hierarchisch aufgebaut sein indem von Knoten auf weitere Diagramme referenziert wird. SA und RT/SA sind im Bereich des Software Engineering (letztere auch im Bereich des SE für eingebettete Systeme) etablierte und bewährte Methoden, die auch zu einem gewissen Maße durch Werkzeuge unterstützt werden.

3.1.2 SDL und GRAPES

SDL (Specification and Description Language) /6/ ist eine von der CCITT standardisierte graphische (und äquivalente textuelle) Sprache zur Spezifikation und Beschreibung von Systemen. SDL ist zwar in erster Linie für den Bereich der Telekommunikation entworfen worden, doch eignet es sich auch für andere Anwendungen. So diskutieren /13/22/ die Anwendung von SDL im Bereich des Elektronikentwurfs, indem SDL in Relation zu VHDL gesetzt wird. SDL hat die Sichtweise von kommunizierenden sequentiellen Prozessen, ähnlich CSP /17/ aber auch VHDL. In SDL wird angenommen, daß es eine Menge nebenläufig aktiver Prozesse gibt, die asynchron über gemeinsame Kanäle durch das Senden und Empfangen von Nachrichten kommunizieren. Dabei impliziert das asynchrone Senden, daß potentiell unbeschränkte Puffer auf den Kanälen angenommen werden müssen. SDL beschreibt ein System in verschiedenen Sichten:

- eine strukturelle Sicht in Form von
  - Block Interaction Diagrams (BD)
- eine Kommunikationssicht in Form von
  - Sequence Charts (SC)
- eine nebenläufige Verhaltenssicht in Form von
  - Process Diagrams (PD).

Die BDs sind nichts anderes als hierarchische Schematics, wie man sie im Bereich des Elektronikentwurfs auf verschiedenen Abstraktionsebenen kennt. Die Kanten in einem BD bedeuten dabei Kommunikationskanäle, wobei zu jedem Kanal eine Menge von darüber zu versendenen Nachrichten gebunden wird. Wie im Fall von RT/SA werden auch hier die anderen Diagramme über Namensgleichheit von Knoten in den verschiedenen Diagrammen angebunden. Die PDs und die CDs beschreiben das dynamische Verhalten. Dabei wird in einem SC die globale Sichtweise angenommen. Es wird beschrieben, welche Kommunikationssequenzen auf den Kanälen beobachtet werden können. Man kann ein SC auch als Protokoll eines Simulationslaufes interpretieren. Folgt man dieser Interpretation so entspricht ein PD der Beschreibung eines Erzeugendensystems für derartige Sequenzen, allerding nur jeweils lokal für einen Knoten des zugeordneten BD. In einem PD wird demnach von der globalen Kommunikationsstruktur abstrahiert indem nur angegeben wird, welche Nachrichten versandt und empfangen werden und wie auf empfangene Nachrichten reagiert wird. PDs werden in Form eines leicht modifizierten Zustandsdiagramms dargestellt.

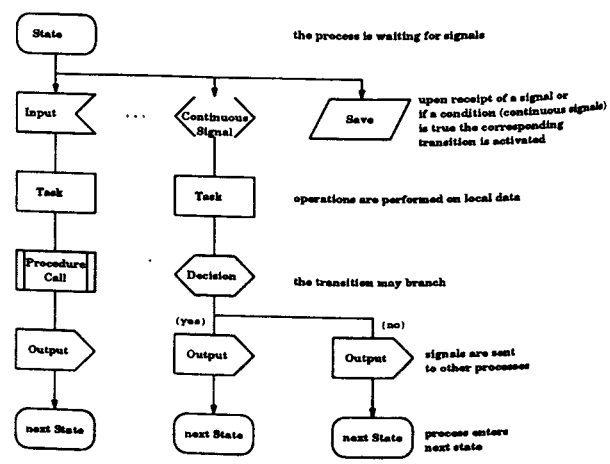

Abb. 2: Beispiel eines SDL Process Diagrams

Dabei geht der Prozeß von einem Zustand in einen Folgezustand über, wenn die hierfür erforderliche Nachricht empfangen wird. Beim Zustandsübergang werden dann mögliche Transformationen durchgeführt und möglicherweise Nachrichten ausgesandt. Je nachdem was spezifiziert ist, werden eingehende Nachrichten, für die kein Zustand, der sie empfangen kann, aktiv ist, entweder abgespeichert oder einfach ignoriert. Neben diesen nachrichtengesteuerten Zustandsübergängen gibt es auch autonome über sogenannte Continuous Signals.
Durch die verschiedenen Sichten erlaubt SDL eine recht umfassende Darstellung eines Systems, die dennoch relativ übersichtlich bleibt. Andererseits jedoch ist es notwendig mehrere Diagramme zu kombinieren, um ein umfassendes Bild von einem System zu erhalten. Dieses Problem tritt noch deutlicher in der von der SNI durchgeführten Weiterentwicklung mit Namen GRAPES auf. GRAPES-86 /16/ ist eine graphische Sprache, die alle Aspekte des Systementwurfs unterstützen soll. Hier werden nun auch die Informationen geliefert, die in SDL-Dokumenten unspezifiziert bleiben. Hier sind besonders die Datenstrukturen zu nennen, die den Nachrichten und prozeßinternen Operationen zugrundeliegen. GRAPES folgt der selben Darstellungsphilosophie wie SDL, d.h. dem Modell der kommunizierenden (sequentiellen) Prozesse. Allerdings erlaubt GRAPES nun auch feinkörnige Parallelität innerhalb der Prozesse. Den BDs in SDL entsprechen in GRAPES Communication Diagrams (CD). Hier wird allerdings ein weiterer Diagrammtyp hinzugebunden, um die Nachrichten, die auf den Kanälen ausgetauscht werden, näher zu spezifizieren. Dies sind die Interface Tables (IT). Wie in SDL wird das lokale dynamische Verhalten der Prozesse mittels PDs beschrieben. Die Syntax und Semantik von GRAPES PDs ist denen von SDL weitgehend gleich. Allerdings führt GRAPES auch synchrone Kommunikation und prozeßlokale, feinkörnige Parallelität. Auch hier werden die unterliegenden Daten weiter spezifiziert, in diesem Fall über Data Tables (DT). ITs und DT referenzieren nur Datentypen. Spezifiziert werden sie in Data Diagrams (DD). Hier werden sowohl Datentypen selbst spezifiziert wie auch deren Beziehung untereinander. Dies wird in Form einer E-R-Notation beschrieben. Um hierarchische Spezifikationen/Modellierungen zu ermöglichen, gibt es noch Hierarchy Diagrams (HD).

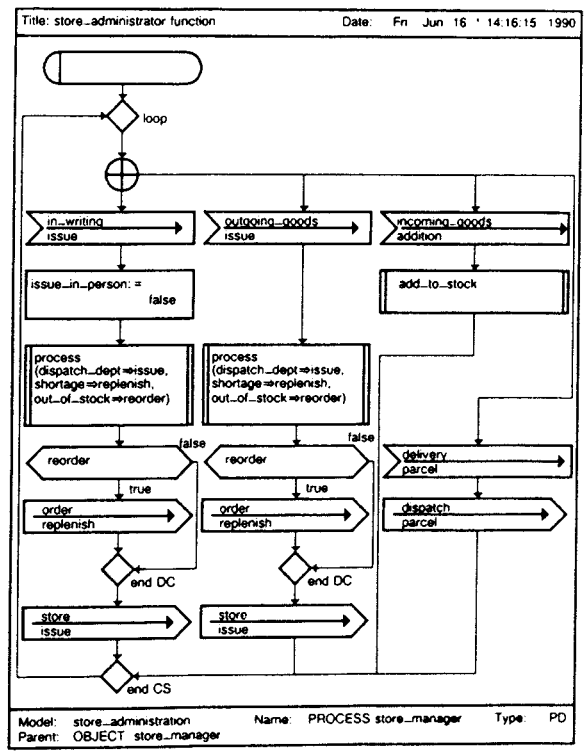

Abb. 3 Beispiel eines GRAPES PD

Man kann GRAPES als einen Versuch ansehen, SA-artige Datenflußdiagramme, SDL-artige Prozeßdiagramme und Datenmodellierung über E-R-Diagramme in ein einziges, syntaktisch und semantisch wohldefiniertes Konzept zu kombinieren. Sowohl für SDL wie auch für GRAPES gibt es aktuelle Versuche, objektorientierte Ansätze zu integrieren. GRAPES hat zudem einen Vorschlag zur Erweiterung von EXPRESS /18/ um Prozeßmodellierung beeinflußt /10/.

3.1.3 StateCharts

StateCharts /15/ sind eine natürliche Fortentwicklung von Zustandsdiagrammen endlicher Automaten (FSM). Um die Komplexität realer Systeme besser beherrschen zu können, werden diese Zustandsdiagramme zunächst um ein Hierarchiekonzept erweitert. Dazu wird erlaubt, daß ein Zustand eine gesamte FSM enhält und daß dieses Konzept rekursiv fortgesetzt werden kann. Graphisch wird in einen Makrozustand ganz einfach die eingebettete FSM gezeichnet. Semantisch ist eine eingebettete FSM aktiv wenn der sie unmittelbar umgebende Makrozustand aktiv ist. Wird so ein Makrozustand aktiviert, so startet die eingebettete FSM von einem Initialzustand, der entweder fest sein kann, oder von der speziellen Aktivierung abhängt. Einen Spezialfall stellen sogenannte History States dar. Dies bedeutet daß eine eingebettete FSM in dem Zustand aktiviert wird, wo sie bei Deaktivierung des umgebenden Makrozustandes war. Auch dieses Konzept läßt sich rekursiv fortsetzen. Ein weiteres Konzept zur Behandlung von Komplexität ist Nebenläufigkeit (divide et impera). StateCharts führen Nebenläufigkeit dadurch ein, daß ein Makrozustand in parallel aktivierbare eingebettete FSMs partitioniert wird. Für nebenläufig aktive Teilautomaten muß ein Synchronisations- und Kommunikationskonzept eingeführt werden. Im Falle der StateCharts bedient man sich hier des Broadcastings. Für StateCharts gibt es leistungsfähige kommerzielle Werkzeuge. Sie werden in der Systemindustrie intensiv eingesetzt. Da es Generatoren gibt, die aus StateCharts qualitativ guten VHDL-Code generieren, ist der Weg zur Elektronikentwicklung geebnet. Zudem gibt es Ansätze /33/, die Konzepte von StateCharts und VHDL zu integrieren.

3.1.4 Petrinetzbasierte Ansätze

Petrinetze werden seit Dekaden für die Spezifikation und Analyse komplexer Systeme eingesetzt. Als Konsequenz verfügt man heute über einen recht umfangreichen Fundus mathematischer Methoden für diesen Ansatz. Petrinetze sind eine graphische Notation für Aktivitätsflüsse, wobei die graphische Darstellung durch Petrinetz Graphen gegeben ist. Dieser besteht aus zwei Typen von Knoten: Stellen zur Darstellung von (lokalen) Bedingungen und Transitionen zur Darstellung von Aktionen. Gerichtete Kanten verbinden Stellen mit Transitionen und Transitionen mit Stellen (bipartiter Graph). Dynamik kommt in dieses Modell durch Markierungen und das Schalten von Transitionen. Stellen können Marken tragen. Eine Transition ist aktiviert, wenn jede ihrer Eingangsstelle mindestens eine Marke trägt. Ist sie aktiviert, so schaltet sie nach endlicher, aber unbestimmter Zeit.
Beim Schalten entfernt sie eine Marke aus jeder ihrer Eingangstellen und fügt jeder ihrer Ausgangsstellen eine Marke hinzu. Man sieht also das lokale Prinzip von Petrinetzen, bei denen das Konzept der Nebenläufigkeit tief verankert ist. Es wurden unterschiedliche Erweiterungen von Petrinetzen untersucht, um dem Problem zu begegnen, daß für reale Probleme Petrinetze dazu tendieren, unübersichtlich zu werden. Zu nennen sind Coloured Petri Nets, bei denen gleichartige Netzmuster übereinandergefaltet werden können und die von Ludmilla Cherkasova definierten Structured Nets /7/. Diese führen ein Hierarchiekonzept ein, das konsistent in Struktur und Verhalten ist. Die Idee ist, in Transitionen Petrinetze einzubetten und dieses Verfahren rekursiv nach innen fortzusetzen. Wenn eine Makrotransition aktiviert wird, so aktiviert sie das eingebettete Netz von seiner initialen Markierung. Wird das eingebettete Netz tot, so schaltet die unmittelbar umgebende Makrotransition.

Eine besonders leistungsfähige Erweiterung stellen die Prädikat/Transitionsnetze (Pr/T-Netze) /12/ dar. Hier sind die Marken Individuen und typisiert. Die Eingangs- und Ausgangskanten von Transitionen sind mit Variablen beschriftet. Eine Transition ist aktiviert, wenn Marken konsistent an diese Variablen gebunden werden können. Transitionen können ein zusätzliches Guardprädikat tragen. Nur wenn dieses Prädikat unter der gefundenen Variablenbindung wahr ist, schaltet die Transition tatsächlich. Beim Schalten kann sie beliebige Transformationen auf Marken vornehmen. Abb. 3 zeigt eine Pr/T-Netz- Transition vor und nach dem Schalten. Pr/T-Netze verbinden Konzepte des Datenflusses mit denen des Kontrollflusses in einem einzigen Diagramm. Durch die Einführung von Hierarchiekonzepten sowohl über Stellen (Analogie zu StateCharts) wie über Transitionen (Anologie zu S-Netzen) /20/ sowie die Einführung von Rekursion /26/ ist ein überaus mächtiges Modellierungskonzept entstanden, das in verschiedenartigen Anwendungen erprobt wurde.

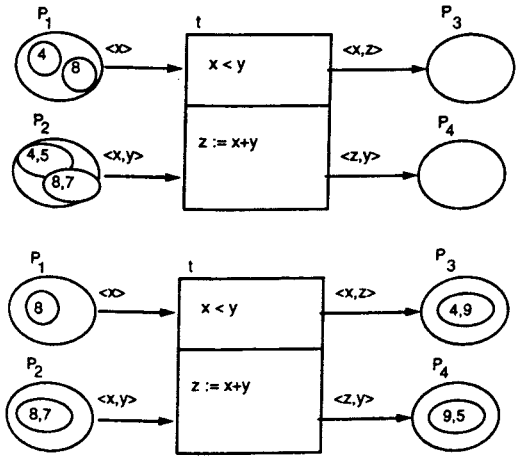

Abb.3: Pr/T-Netze vor und nach dem Schalten

3.2 Hardwarebeschreibungssprachen

Hardwarebeschreibungssprachen wie VHDL wurden entwickelt, um bestimmte Bereiche des Entwurfs digitaler Hardware abzudecken. Es sollte jedoch erwähnt werden, daß Bestrebungen bestehen, auch ein analoges VHDL zu entwerfen. Da zur Systemmodelierung ein multiparadigmatischer Ansatz anzustreben ist, ist CHDLs wie VHDL eine spezielle Rolle zuzuweisen, die von diesen Sprachen besonders effizient erbracht werden kann. Die Stärke liegt eindeutig bei digitaler Hardware im Bereich algorithmische Ebene bis Gatterebene. Einige (allerdings inzwischen historisch zu nennende Ansätze) wie DACAPO /8/ deckten auch die Systemebene des Elektronikentwurfs mit ab. Zu diesem Zweck unterstützt DACAPO Konzepte wie Abstrakte Datentypen, Module, Monitore, Interrupts, feinkörnige Nebenläufigkeit innerhalb von Prozessen, alles Konzepte, die in VHDL leider nicht verfügbar sind. Durch Einbettung von VHDL in mächtigere Beschreibungsmittel lassen sich diese Defekte möglicherweise beheben. Der Ansatz der SpecCharts /33/ weist in die richtige Richtung.

3.3 Axiomatische Ansätze

Alle bisher diskutierten Spezifikations- und Modellierungsmethoden folgem einem operationalen Paradigma. Will man von einem angenommenen Interpreter völlig abstrahieren, landet man bei axiomatischen bzw. algebraischen Ansätzen. Das wohl bekannteste Beispiel für diesen Ansatz ist die Spezifikationssprache Z /28/. Diese Sprache ist vom BSI standardisiert und die ISO-Standardisierung ist eingeleitet. Z folgt der üblichen mathematischen Notation der Mengenlehre (der Name ist ein Sprachspiel mit dem Wort "set"). Das entscheidende Konzept ist die Einführung von Zuständen und Zustandsübergängen. Damit wird es

möglich, Dynamik in Spezifikationen auszudrücken, was schwierig wäre, wenn man nur Mengen, Relationen und Funktionen zur Verfügung hätte. Sehr interessant an Z ist auch die Tatsache, daß alle Spezifikationen in Form spezieller Schemata aufgeschrieben werden und es auf diesen Schemata einen algebraischen Kalkül gibt. Z wurde erfolgreich in verschiedenen Hardware- und Softwareprojekten (z.B. Transputer T800) erfolgreich eingesetzt. Es existiert eine umfangreiche Literatur zu dieser Spezifikationsmethode.

Noch eleganter ist der Ansatz der Dynamischen Algebren /14/. Auch hier gibt es einen Zustandsbegriff und Zustandsübergänge. Dadurch aber, daß Zustände durch vollständige Algebren und Zustandsübergänge durch Erweiterungen dieser Algebren dargestellt werden, erhält man ein überaus mächtiges Werkzeug. Im Gegensatz zu den konzeptionell verwandten Ansätzen der Denotationalen Semantik müssen allerdings bei Zustandsübergängen nur die tatsächlichen Änderungen spezifiziert werden. Dynamische Algebren wurden erfolgreich zur formalen Spezifikation so unterschiedlicher Objekte wie PROLOG, PVM /3/ und VHDL /4/ erfolgreich eingesetzt.

4 Methoden und Werkzeuge des integrierten Systementwurfs

4.1 Systemsimulation

Die Gesamtspezifikation des zu bauenden Systems ist im Idealfall einer "top down"-Vorgehensweise das erste formale Dokument, das zur Verfügung steht. Es ist daher formal höchstens auf interne Konsistenz überprüfbar, keinesfalls auf Konsistenz mit den Intentionen des Entwicklers. Simulation scheint daher der einzig mögliche Ansatz zu sein, um den Entwerfer die Chance zu geben, zu prüfen, of die nun ausführbar gewordene Spezifikation seinen Intentionen entspricht. Man kann sicherlich über den Wert der Simulation auf jeder Abstraktionsebene streiten, die Simulation des initialen Dokuments ist durch nichts ersetzbar.

Die Frage ist, welches die adäquate Simulationstechnik ist. Bedingt durch die heterogene Natur von Systemen schließt sich ein monolithisches Simulationssystem von selbst aus. Als Lösung bieten sich Multisimulatorsysteme (Simulation Backplanes) an. Hier können spezielle dedizierte Simulatoren Teilaspekte effizient nachbilden und nur die Interaktionen werden von der Backplane verwaltet. Dabei sind drei Hauptprobleme zu lösen:

- Datenaustausch zwischen den Simulatoren,
- Synchronisation der verschiedenen Simulatoren,
- Homogenisierung der Benutzerschnittstelle

Davon erscheint das Synchronisationsproblem das zentrale zu sein. Das Problem besteht darin, die einzelnen Simulatoren hinreichend synchron zu halten, so daß bei jedem Datenaustausch zwischen Simulatoren der Zeitpunkt für alle beteiligten Simulatoren legal, d.h. nicht in der Vergangenheit liegend ist. Hierzu gibt es zwei extreme Lösungen:

- der (übersynchronisierende, pessimistische) Supervisoransatz,
- die (untersynchronisierende, optimistische) Time Warp Methode.

Der Supervisoransatz als pessimistischer Ansatz erlaubt stets nur dem Simulator voranzugehen, der in der nächsten relativen Zukunft eine Aktion plant. Der Ansatz ist sicher, erlaubt aber keinerlei Nebenläufigkeit, was dazu führt, daß insb. auf Parallelrechnern eine schlechte Performanz erzielt wird. Beim Time Warp Ansatz /19/

wird optimistisch angenommen, daß alle Simulatoren völlig unabhängig laufen können, ohne daß sich illegale Situationen ergeben. Diese Annahme stimmt, solange kein Simulator versucht in die Vergangenheit eines anderen Simulators zu senden. Sollte dies geschehen, muß der empfangende Simulator auf den Empfangszeitpunkt zurückgesetzt werden. Da er seinerseits in der Zwischenzeit möglicherweise Nachrichten versandt hat, müssen diese mittels "Antinachrichten" anulliert werden, d.h. weitere Simulatoren müssen zurückgesetzt werden.
In jüngerer Zeit wurden sehr anspruchsvolle Ansätze der Multisimulation veröffentlicht, z.B. /24/. Die CFI arbeitet derzeit an einem Standard für eine Simulation Backplane.
Der Simulator selbst ist nur ein Hilfsmittel, um Experimente durchzuführen. In der Experimentplanung, -Durchführung und -Auswertung liegt die eigentliche Aufgabe. Um diese Aufgabe zu unterstützen bieten sich KI-Techniken an. Dies wurde von verschiedenen Autoren untersucht, besonders vielversprechend von /25/.

4.2 Systemanalyse und Partitionierung

Ist der Entwerfer auf der Basis der vorgenommenen Simulationen überzeugt, daß die vorliegende Spezifikation seinen Intentionen entspricht, kann der eigentliche Entwurfsprozeß beginnen. Grundlage ist eine eingehende Analyse des vorliegenden Gesamtmodells. Dabei kann die Analyse sowohl auf den erzielten Simulationsergebnissen basieren, als auch statischer Natur sein. Ein Beispiel einer derartigen Analyse, die gut verstanden ist, und wofür leistungsfähige kommerzielle Werzeuge verfügbar sind, ist die Leistungsanalyse. Wann immer es gelingt, ein System als Warteschlangenmodell adäquat darzustellen, können Ergebnisse der Warteschlangentheorie für analytische oder in komplizierteren Fällen simulative Analysen genutzt werden. Ein anderes Beispiel ist die Testbarkeitsanalyse. Dieses Gebiet ist auf niedrigeren Abstraktionsebenen des Elektronikentwurfs gut verstanden. Neuere Arbeiten behandeln den Testbarkeitsaspekt bereits auf der algorithmischen Ebene /23/ und auch regelbasierte Ansätze werden verfolgt /2/. Mit Methoden der Netzplantechnik oder der aus dem Hardwarentwurf entstandenen Timing Analysis können kritische Pfade in nebenläufigen Algorithmen und Abschätzungen des Zeitbedarfs auf diesen kritischen Pfaden ermittelt werden. Mit ähnlichen Methoden wird in Realzeitsoftware eine Schedulability Analysis vorgenommen.
Für den integrierten Systementwurf sind jedoch solche Analysen von besonderer Bedeutung, die eine Partitionierung des Systems in Teilsysteme determiniert, wobei hier zunächst Teilsysteme in verschiedenen Ingenieursdomänen gemeint sind. Innerhalb der Domänen kann dann eine weitere Partitionierung stattfinden. Es kann angenommen werden, daß sich Fragen der Partitionierung in ein Informationstechnik-Teilsystem und die Umgebung dieses IT-Systems durch naheliegende Überlegungen lösen lassen, d.h. nicht durch ausgefeilte Analysetechniken zu unterstützen sind. Eine naheliegende Vorgehensweise ist, alle Energieströme auf inhärenten Informationstransport zu analysieren und zu untersuchen, ob das mechanische System signifikant einfacher wird, wenn es sich auf den Energietransport beschränken kann. In den meisten Fällen ist die an den Informationstransport und die Informationsverarbeitung gekoppelte Aktivität (meist Steuern und Regeln) billiger und effizienter durch ein eingebettets IT- Subsystem zu lösen. Man vergleiche nur die (bewundernswerte) Mechanik eines analogen Plattenspielers mit der eines CD-Players. Ist das IT-Subsystem identifiziert und liegen seine Schnittstellen nach außen fest, so gilt es, dieses selbst zu analysieren und weiter zu partitionieren. Hierzu ist es hilfreich, das IT-Gesamtsystem

zunächst als Multiagentensystem aufzufassen, d.h. den vorliegenden (nebenläufigen) Algorithmus in solche Cluster aufzuteilen, die untereinander möglichst schmalbandig gekoppelt werden können. Für eingebettete Systeme erscheint wohl das Paradigma der botschaftengekoppelten MIMD-Systeme am ehesten adäquat zu sein. Hat man die lokalen Agenten identifiziert, gilt es, deren interne Realisierung festzulegen. Dabei ist sowohl an die Bedienung der Schnittstellen zur Umgebung, wie auch an die internen Abläufe zu denken. Grundsätzlich ist ein lokaler Verarbeitungsknoten stets durch Programmierung eines universellen, programmierbaren Bausteins (Signal- oder Universalprozessor) zu realisieren, falls nicht spezielle Restriktionen (Zeit, Kosten bei extrem hohen Serien, Stromverbrauch) dagegen sprechen. Durch Faltung verschiedener Teilalgorithmen, die in einem lokalen Knoten ablaufen, auf einen Prozessor kann die Kosteneffizienz weiter gesteigert werden. Voraussetzung ist die Verfügbarkeit eines leistungsfähigen Realzeitschedulers und ein positives Ergebnis der Schedulability Analysis. Diese Faltung ist zumindest erheblich schwieriger, wenn man sich für eine dedizierte Hardware für die zu realisierenden Algorithmen entscheidet. Einen interessanten Kompromiß stellen Konfigurationen dar, die aus einem programmierbaren Prozessor und daran angeschlossene Coprozessoren bestehen. Dieser Ansatz wird in jüngerer Zeit intensiv unter dem Begriff Hardware/Software Codesign untersucht. Natürlich ist auch hier wieder die Frage der Partitionierung in auf dem programmierbaren Prozessor laufende Software und Spezialhardware zu lösen. Darauf soll unter 4.3.1 kurz eingegangen werden.

4.3 Syntheseaktivitäten

Unter Synthese versteht man im Entwurfsprozeß die Überführung eines abstrakteren Modells (einer Spezifikation) in ein weniger abstraktes. Als Beispiel kann die Logiksynthese dienen, die eine Spezifikation auf der RT-Ebene in ein Modell auf der Gatterebene überführt, oder die High-Level Synthese, welche von algorithmischen Spezifikationen eine RT-Realisierung synthetisiert. Traditionell steht einem Syntheseverfahren auf der Zielebene eine Bibliothek benutzbarer Elementarbausteine zur Verfügung. Das Syntheseergebnis ist dann eine Menge von Instantiierungen als Auswahl aus dieser Bibliothek und eine Verschaltung dieser Instantiierungen. Dieses Bibliothekskonzept läßt sich jedoch erheblich virtualisieren. An die Stelle fester oder in Maßen generischer Bibliothekselemente können auch Generierungsalgorithmen treten. So wie das Suchen von speziellen Bausteinen in umfangreichen Bibliotheken durch eine Vielzahl von Suchbegriffen gesteuert werden muß, kann mit vergleichbarem Aufwand auch ein Generierungsalgorithmus parametrisiert werden. Der Aufwand für den Benutzer ist also vergleichbar, der Nutzen auf der Seite der Elementanbieter aber enorm. Statt umfangreiche Bibliotheken erstellen und pflegen zu müssen, fällt nur noch die Entwicklung eines zugegebenermaßen komplexen Generators an, eines Generators dessen Ergebnisse ebenso stabil sein müssen wie heutige Bibliothekselemente. Der Generatoransitz steckt noch in den Kinderschuhen, erscheint aber für eine konsequente Verfolgung des HW/SW-Codesign essentiell zu sein, da nur so eine homogene Behandlung von Hardwaregenerierung und Codeerzeugung möglich ist. Beispielse für ein generatives Vorgehen im Logikentwurf ist /5/ und im Layoutentwurf /30/.

4.3.1 Hardware/Software Codesign

HW/SW-Codesign kann man als eine konsequente Weiterentwicklung der High-Level Synthese verstehen. Gilt es bei der traditionellen HL-Synthese, einen Algorithmus in eine Verschaltung von RT-Elementen um-

zusetzen, so wird hier eine spezielle Komponente, nämlich ein frei programmierbarer Prozessor hinzugenommen, auf dem Teile des zu implementierenden Algorithmus als Software realisiert ist. Die Hauptprobleme, welche vom HW/SW-Codesign zu lösen sind, sind die Partitionierung in HW und SW und die Generierung geigneter Schnittstellen zwischen den Teilen. Dabei konzentriert man sich derzeit i.d.R. auf Konfigurationen, die aus einem Universalprozessor und einem oder mehreren daran angeschlossenen dedizierten Coprozessoren (über Coprozessorschnittstelle oder als DMA-Komponenten am Systembus) beschränken. Die zu verarbeitende Spezifikation für HW/SW-Codesign muß neben der intendierten Funktionalität in Form eines zu realisierenden Algorithmus auch Restriktionen, insb. bzgl. des Timing beinhalten. Basierend auf dieser Information kann geprüft werden, ob eine reine Softwarelösung in Form paralleler Threads den Restriktionen genügt. Wenn nicht, muß analysiert werden (mit ähnlichen Methoden wie bei der HL-Synthese) welche Algorithmenteile (Threads) durch dedizierte HW realisiert werden können. Dabei muß insb. auch der Overhead, der durch den Datenaustausch auf der Schnittstelle zwischen Prozessor und Coprozessor und der Ansteuerung des Coprozessors entsteht, berücksichtigt werden. Alternativ kann auch von einer reinen HW-Lösung ausgegangen werden, bei der bei Verstoß gegen Restriktionen (z.B. Fläche) Teile herausgelöst und in Software verlagert werden. Eine Analogie zu den oben betrachteten Threads sind hier Sequenzen im Daten-/Kontrollflußgraph, wobei sich derartige Sequenzen bzgl. ihrer Kosten nach verschiedenen Kriterien (Komplexität der benötigten Operationswerke, Sequentialität,...) bewerten lassen. Hier lassen sich Methoden, die bei der HL-Synthese zur Bestimmung optimaler Schedules und Allokationen benutzt werden, übertragen. Natürlich ist auch bei diesem Vorgehen zu berücksichtigen, daß sich durch das einzuziehende Interface zusätzliche Kosten ergeben.

4.3.2 Eingebettete Software Systeme

Als Alternative zu in Systeme eingebetter Spezialelektronik ist stets auch der Rückgriff auf Standardprozessoren, entweder in Form von Universal- oder von Signalprozessoren und darauf laufender Software zu sehen. Durch die hohe Serie derartiger Prozessoren wird ein sehr geringer Stückpreis erreicht und dadurch, daß wegen der großen Serie auf "leading edge" Technologie zurückgegriffen werden kann, wird auch eine enorme Leistungsfähigkeit erreicht. Zudem birgt eine reine Softwarelösung eine enorme Flexibilität in sich. Zwar kann man argumentieren, daß zwischen dem Aufwand einer HL-Synthese und dem eines optimierenden Codegenerators für einen Standardprozessor grundsätzlich kein zu großer Unterschied besteht. Die spezielle Fertigung eines ASICs jedoch steht in keinem Verhältnis zu den Reprogrammierungskosten eines Standardprozessors. Es darf aber nicht übersehen werden, daß der reine Softwareansatz ebenfalls enorme Entwurfsprobleme mit sich bringt. Letztlich gilt es, eine Realzeitprogrammierumgebung zu realisieren. Die zentrale Komponente dabei ist ein Realzeitscheduler. Die erforderliche Flexibilität erhält er durch ein Multilevel Scheduling. Ob eine bestimmte Problemstellung überhaupt auf einem Prozessor realisierbar ist, läßt sich mittels Schedulability Analysis feststellen. Im Verletzungsfall kann man versuchen, auf Multiprozessorsysteme auszuweichen. Hier stellen sich dann aber Probleme des Realzeitroutings und des Realzeitschedulings auf parallelen Prozessoren, für die es derzeit noch keine abschließenden Verfahren gibt. Es ist aber zu erwarten, daß die Entwicklung genau in die Richtung auf eingebettete Software, die auf botschaftengekoppelten Multiprozessorsystemen läuft, gehen wird. Ob-

jektorientierte, auf Microkernels aufsetzende Ansätze /1/27/ sind vielversprechende Ansätze in diese Richtung.

4.4 Concurrent Engineering

Verfolgt man den integrierten Systementwurf, so müssen alle Aspekt im Laufe des Lebenszyklus eines Produkts berücksichtigt werden. Diese Kohärente Betrachtung all dieser Interdependezen wird Concurrent Engineering (CE) genannt /29/. Zu betrachtende Aspekte beinhalten dabei Funktionalität, Leistung, Preis, Preis/Leistungs-Verhältnis, Wartbarkeit, Fabrizierbarkeit, Marketingaspekte, Ersetzbarkeit von Vorprodukten, Managementkosten, Entsorgungsaspekte, rechtliche Gesichtspunkte, etc. Vom US Institute for Defense Analysis wurde Concurrent Engineering definiert als:
"Concurrent Engineering is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developer, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule and user requirements".
All die einzelnen Aspekte sehen nur einen kleinen Auschnitt des Gesamtproblems und versuchen in diesem Rahmen zu optimieren. Um ein globales Optimum zu finden, müssen die Einflüsse anderer Aspekte berücksichtigt werden, wobei die entscheidenden internen Prozesse der beeinflussenden Aktivitäten meist nicht kennt (kennen muß). Concurrent Engineering muß nun kontinuierlich die notwendige Information an die Agenten liefern, die spezielle Aspekte behandeln und auf der Basis eines umfassenden Entwurfsmodells versuchen, ein globales Optimum als Kompromiß der lokalen Optima zu finden. Damit sind aber neue Komplexitäten im Entwurfsprozeß verbunden, die nur mit Rechnerunterstützung gelöst werden können (Computer Aided Concurrent Engineering, CACE). Zwei Hauptbereiche von CACE können identifiziert werden: Unterstützung der Nebenläufigkeit und Verfolgen der verschiedenen, unabhängigen Aspekte des Entwurfs. Um die Kooperation verschiedener Entwurfsteams zu unterstützen, muß Computer Supported Cooperative Work (CSCW) ein integraler Bestand von CACE sein.
Die Grundlage eines CACE-Systems bildet ein leistungsfähiges CACE- Framework. Frameworkkonzepte wurden und werden von der CFI (CAD Framework Initiative), einer internationalen Vereinigung von über 50 Firmen und Organisationen standardisiert. CFI-kompatible Werkzeuge können in nach CFI-Standard konstruierte Frameworks relativ einfach integegriert werden. Dabei sind drei verschiedene Integrationsdimensionen zu unterscheiden:

- Datenintegration,
- Kontrollintegration,
- Benutzungsoberflächenintegration.

Zusätzliche Komponenten vervollständigen die Infrastruktur, die lt. CFI- Standard von einem Framework anzubieten sind:

- Methodenmanagement,
- Benutzungsschnittstellenmanagement,
- einheitliche Designdarstellung basierend auf einem Datenmanagement.

Eine besondere Rolle, besonders bei CE-Anwendungen, spielt das Designflowmanagement /21/35/. Es verwaltet all die während des Entwurfsprozesses entstehenden Dokumente incl. der verschiedenen Versionen, steuert die sequentielle oder parallele Aktivierung von Werkzeugen, macht Vorschläge bzgl. der Auswahl anwendbarer Werkzeuge und produziert all die Berichte,

durch die der Entwurfsprozeß nachvollziehbar wird.
Mit diesen Diensten können nun leistungsfähige Entwurfsumgebungen konstruiert werden. Ein besonders fortschrittliches Beispiel eines für CE-Anwendungen sehr gut geeigneten Framworks ist das JESSI Common Framework (JCF) /31/, das unter dem Namen SIFRAME von der SNI als CACE-Framework angeboten wird. Ein aktueller Trend geht nun in Richtung auf föderierte Frameworks. Man trägt damit der Tatsache Rechnung, daß es i.d.R. für die durch CE zu integrierenden Domänen bereits frameworkbasierte Umgebungen gibt. Mit Föderierungsdiesnsten wird es ermöglicht, die Interdependenzen zwischen diesen Umgebungen zu organisieren, ohne die Autonomie der Einzelumgebungen zu sehr einzuschränken.

5 Zusammenfassung

Der integrierte Systementwuf ist eine neue Herausforderung an Wissenschaft un Industrie. Besonders in Deutschland, das wie kein anderes Land vom Export hochwertiger, hochkomplexer Waren abhängig ist, spielt dieser Ansatz eine zentrale Rolle. Durch die integrierte und möglichst nebenläufige Behandlung aller an der Entstehung eines Produkts beteiligten Aktivitäten können in kürzerer Zeit und zu günstigeren Preisen qualitativ hochwertige Produkte entstehen.
In diesem Beitrag wurde versucht, darzustellen, wie bekannte Ingenieursverfahren in Richtung integrierter Systementwurf weiterentwickelt werden können. Dabei wurde zunächst die Bedeutung der Modellierung erläutert und gezeigt, wie sich bekannte Modellierungsverfahren kombinieren lassen, um zumindest rechnerintern eine Gesamtmodellierung anbieten zu können. Derartige Gesamtmodelle bilden dann den Rahmen für Entwurfsaktivitäten, wobei hier dafür plädiert wurde, weitgehend auf bewährte Methoden der beteiligten Ingenieursdomänen zurückzugreifen. Zusätzlich ist jedoch eine Partitionierung in Teilkomponenten, die diesen Domänen zugeordnet werden können, vorzunehmen und die Querbezüge zwischen Entwurfsaktivitäten müssen verwaltet werden. Die Rolle, die dabei eine rechnergestützte Infrastruktur, ein CACE-Framework spielt, wurde am Ende des Beitrages kurz dargestellt.

Zusammenfassend kann gesagt werden, daß die notwendige Basistechnologie vorhanden ist, sodaß es nur eine Frage der Zeit sein dürfte, bis sich der integrierte Systementwurf auf breitester Front durchgesetzt haben wird.

6 Literatur

/1/ R. Berg, J. Cordsen, J. Heuer, J. Nolte, B. Oestmann, M. Sander, F. Schön, W. Schröder-Preikschat: The PEACE Family of Operating Systems. Techn. Report GMD FIRST, 1991

/2/ M. Bidjan-Irani: A Rule-Based Design for Testability Rule Checker. IEEE Design & Test of Computers, March 1991

/3/ E. Börger, U. Glässer: A formal Specificaiton of the PVM architecture. Proc. IFIP Congress'94, Elsevier, 1994

/4/ E. Börger, U. Glässer, W. Müller: The Formal Semantics of Behavioral VHDL 92 Descriptions. Proc. EURO-VHDL'94, 1994

/5/ F. Buijs: Automating the Logic Synthesis of Arithmetic Logic Units. Diss. Univ.-GH Paderborn, 1994

/6/ CCITT Recommendation Z.100: Specification and Description Language SDL. AP IX-35, 1988

/7/ L.A. Cherkasova, V.E. Kotov: Structured Nets. Proc. MFCS'81, Springer LNCS 118, 1981

/8/ DACAPO III Systems User Manual. Dosis GmbH, Dortmund, 1987

/9/ T. DeMarco: Structured Analysis and Systems Specification. Prentice Hall, 1978

/10/ W. Felser, W. Müller: EXPRESS-P - Eine Erweiterung von ISO 10303-11 zur Verhaltensmodellierung. Tagungsband GI CAD'94, 1994

/11/ D.D. Gajski: The Structure of a Silicon Compiler. Proc. IEEE ICCD, 1987

/12/ H.J. Genrich, K. Lautenbach: System Modelling with High-Level Petri Nets. Theoretical Computer Science, 13, 1981

/13/ W. Glunz, G. Venzl: Hardware Design Using CASE Tools.
Proc. IFIP VLSI'91, 1991

/14/ Y. Gurevich: Evolving Algebras - A Tutorial Introduction. Bull. of the EATCS, Feb. 1991, No. 43

/15/ D. Harel: StateCharts: A visual formalism for complex systems. Science of computer Programming, 8, 1987

/16/ G. Held (Ed.): Sprachbeschreibung GRAPES. Siemens AG, 1990

/17/ C.A.R. Hoare: Communicating Sequential Processes. Comm. ACM, Vol. 21, No. 8, 1978

/18/ ISO/DIS 10303-11: EXPRESS Language Reference Manual. ISO TC 184/SC4, August 1992

/19/ D.R. Jefferson, H.A. Scwizral: Fast concurrent Simulation using the time warp mechanism. Proc. SCS Distributed Simulation Conference, 1985

/20/ Elisabeth Kleinjohann: Integrierte Entwurfsberatung auf der Basis erweiterter Prädikat-Transitionsnetze. Diss. Univ.-GH Paderborn, 1994

/21/ E. Kupitz: Design Assistance in Concurrent Integrated Environments. Proc. IFIP EDAF'92, North Holland, 1992

/22/ B. Lutter, W. Glunz, F.J. Rammig: Using VHDL for Simulation of SDL Specifications. Proc. EURODAC'92, 1992

/23/ Ch. Nagel: Synthesis for Testability by Synthesis Controlling. Proc. EUROMICRO'93, North Holland, 1993

/24/ M. Niemeyer: Simulation of Heterogeneous Models with a Simulator Coupling System. Proc. SCS 1991 European Simulation Multiconference, June 1991

/25/ H. Pfaffhausen: Ein wissensbasierter Ansatz zur automatischen Durchführung von Experimenten in der Logiksimulation. Diss. Univ.-GH Paderborn, 1991

/26/ F.J. Rammig: System Level Design. J.P. Mermet (Ed.): Fundamentals and Standards in Hardware Description Languages, Kluwer, 1993

/27/ U. Rozek: TINIX - ein verteiltes Betriebssystem für Transputer. Zeitschrift IX, (3), 1992

/28/ J.M. Spivey: The Z Notation: A Reference Manual. Prentice Hall

/29/ R.A. Sprague, K.J. Singh, R.T. Wood: Concurrent Engineering in Product Development. IEEE Design & Test of Computers, March 1991

/30/ W. Spruth: The Design of a Microprocessor. Springer, 1989

/31/ B. Steinmüller: The JESSI COMMON FRAME Project - A Project Overview. Proc. IFIP EDAF'92, North Holland, 1992

/32/ T. Tikkanen, T. Leppnen, J. Knivel: Structured Analysis and VHDL in Embedded ASIC Design and Verification. Proc. EDAC'90, 1990

/33/ F. Vahid, S. Narayan, D.D. Gajski: SpecCharts: A Language for System Level Synthesis. Proc. IFIP CHDL 91, North Holland, 1991

/34/ P.T. Ward, S.J. Mellar: Structured Development for Real-Time Systems, Vol. 1-3, Yourdan Press, N.Y., 1985

/35/ M. Zanella: Principals of Design Methodology Management for Electronic CAD Frameworks. Proc. EDAC'92, 1992