# Mehrebenensimulation

Franz J. Rammig
Universität - GH - Paderborn
Fachbereich Mathematik/Informatik

## 1) Einleitung

In diesem Beitrag sollen einige Tendenzen der Mehrebenensimulation [RAM87] aufgezeigt werden. Da dies kaum ohne Betrachtung der Simulation auf einzelne Abstraktionsebenen möglich ist, werden diese Simulationstechniken zuvor kurz eingeführt.

Grundsätzlich bedeutet Simulation [FIS78, ZEI76] , ein System zu konstruieren, das sich wie ein anderes verhält. In unserem Kontext bedeutet es, ein Model des betrachteten Entwurfsobjekts auszuführen. D.h. durch Simulation sorgt man dafür, daß sich ein Gastrechner wie das zu simulierende Objekt verhält. Es ist weit verbreitet, Simulation als Verifikationswerkzeug zu betrachten. Jedoch ist die Simulation nur ein Datenproduzent für eine Verifikation. Die eigentliche Verifikation wird durch Untersuchung der durch den Simulator produzierten Daten durchgeführt. Dabei ist zu beachten, daß, wenn nicht autonome Systeme simuliert werden, die Simulationsergebnisse nicht nur von dem auszuführenden Modell abhängen, sondern auch von den Daten, die diesem Modell von der Umgebung angeboten werden.
Somit bedeutet Verifikation auf der Basis von Simulation letztendlich, zu analysieren, ob das Paar (Modell des Entwurfsobjekts, Daten von der Umgebung) die korrekten Ergebnisdaten produziert. Demnach besteht ein komplettes Verifikationssystem auf der Basis von Simulation aus vier Hauptkomponenten:

- Ein Generator für ausführbare Modelle von Objektumgebungen,
- ein Generator für ausführbare Modelle der zu simulierenden Objekte,
- ein Laufzeitsystem für das Modell zusammen mit seiner Umgebung und
- ein Ergebnisanalysator

Die Glaubwürdigkeit einer Verifikation, die auf der Basis von Simulation durchgeführt wird, hängt von allen beteiligten Komponenten und Modellen ab: Zunächst hat man nur ein Modell eines Objekts. Alle Verifikationsaussagen, die durch das Simulationssystem gewonnen werden, sind Aussagen über dieses Modell. Modelliert es das reale System schlecht, haben die Aussagen wenig mit diesem zu tun. Hier haben die Modellierungsmöglichkeiten der benutzten Hardwarebeschreibungssprachen zwar einen gewissen Einfluß (man kann nie präziser modellieren, als es das benutzte Modellierungswerkzeug zuläßt), aber zumindest die Qualität des initialen Modells innerhalb eines Entwurfsprozesses hängt weitgehend vom Modellierungsgeschick des Entwerfers ab.
Da ein Simulationslauf nur Ergebnisse in Bezug auf die Eingabe, mit der er konfrontiert wird, produziert, hängt die Aussagekraft einer Verifikationsaussage wesentlich von diesen Eingabemustern ab. Somit ist ein Modell der Umgebung, das so präzise und vollständig wie möglich ist, wesentlich. Das ausführbare kombinierte Modell des zu simulierenden Objekts und seiner Umgebung muß nun durch ein Laufzeitsystem ausgeführt werden, das das Modell auf Abläufe auf einem Gastrechner abbildet. Diese Abbildung kann von sehr unterschiedlicher Qualität sein, mit der üblichen Balance zwischen Geschwindigkeit und Präzision.

Endlich sind die produzierten Daten nicht mehr wert, als die darauf angestellte Analyse. Hier ist das kritischte Problem zu suchen. Da in den meisten Fällen die Analyse manuell auf einer enormen Datenmenge, die zudem oft in "anti-ergonomischer" Weise dargestellt werden muß, ist die Wahrscheinlichkeit inkorrekter Schlußfolgerungen relativ hoch. Dies ist besonders gefährlich, da der Entwerfer ein gutes Gefühl hat, weil er das Entwurfsobjekt sorgfältig simuliert hat und keine weiteren Fehler entdeckt hat. Rechnerunterstützte Ergebnisanalysesysteme sind daher ein wichtiger Forschungsgegenstand.

# 2) Abstraktionsebenen und Sichten

Unter einem Mehrebenensimulator soll ein Simulator verstanden werden, der mindestens zwei verschiedene Abstraktionsebenen überdeckt. Dabei ist im Sinne von Gajski's Y-Diagramm [GAJ87] für die Simulation nur die Verhaltenssicht von Bedeutung. Es ist ja gerade das Verhalten eines elektronischen Systems, was simuliert werden soll. Die Struktursicht mag dazu dienen, ein Verhaltensmodell aus dem Verhalten der Einzelkomponenten zusammenzusetzen und die Geometriesicht mag Information darüber liefern, wie Zeitparameter präzise eingestellt werden müssen, zur Ausführung endlich kommt ein Verhaltensmodell. Daher sollen hier die verschiedenen Abstraktionsebenen auch nur eingeschränkt auf die Verhaltenssicht diskutiert werden [RAM89].

## Systemebene

Auf dieser Ebene wird das zu modellierende Objekt als System kooperierender semiautonomer Module wie Prozessoren, Kanäle, etc. gesehen, wobei all diese Komponenten als Prozessoren im weiteren Sinn interpretiert werden, also als Einheiten, die in wohldefinierter Weise auf Instruktionen reagieren. Um auf dieser Ebene Systeme zu modellieren, muß pro beteiligter Komponente die Syntax und die Semantik des "Instruktionssatzes" und globald die Syntax und Semantik der Kommunikationsstruktur (Protokolle) definiert werden.

## Algorithmische Ebene

Die auf der Systemebene angegebenen Instuktionssätze müssen von jeder Komponente interpretiert werden. Diese Interpretationsalgorithmen werden auf der algorithmischen Ebene in imperativer Form spezifiziert. In der Regel sind diese Interpretationsalgorithmen hochgradig nebenläufig.

## Registertransferebene

Diese Ebene ergibt sich aus der algorithmischen Ebene durch Inversion der Sichtweise. Betrachtet man auf der algorithmischen Ebene ein System aus der Sicht der Steuerung (imperativ) so wird es auf der RT-Ebene von den beteiligten Einzelkomponenten her gesehen (reaktiv). Solch eine Komponente ist passiv, bis ein bestimmtes Ereignis eintritt. In diesem Fall reagiert sie in wohldefinierter Weise, um danach wieder passiv zu werden.

## Gatterebene

Auf der Gatterebene wird die Unterscheidung zwischen Steuerleitungen und Datensignalen aufgegeben. Hier hat man die Vorstellung eines (Booleschen) Gleichungssystems. Dieses Gleichungssystem kann durch externe Stimuli oder interne Instabilitäten detabilisiert werden. Als Reaktion darauf wird es jedoch stets versuchen, wieder einen stabilen Zustand (eine Lösung) einzunehmen.

## Schalterebene

Auf der Schalterebene hat man grundsätzlich das gleiche Modellierungskonzept wie auf der Gatterebene: stimulierte Gleichungssysteme. Hier werden nun aber Systeme ohne a priori vorgegebene Sig-

nalflußrichtung betrachtet und die Einzelkomponenten sind Schalter und ladungstragende Knoten. Diese Fähigkeit der Knoten, Ladung zu tragen, bringt es mit sich, daß der Wertbereich von Variablen auf dieser Ebene auf verschiedene Signalstärken ausgedehnt werden muß. Sind die Gleichungen bereits auf der Gatterebene nicht mehr Boolesch, da man beispielsweise unsichere Werte und Signalübergänge mitmodellieren möchte, so sind Boolesche Gleichungssysteme auf der Schalterebene völlig ausgeschlossen.

### Elektrische Ebene

Beschreibt man das detaillierte Verhalten der elektrischen Basiskomponenten in Form von Differentialgleichungssystemen, so hat man die elektrische Ebene erreicht. Das Grundmodell der stimulierten Gleichungssysteme, das auch auf der Gatter- und Schalterebene benutzt wird, ist auch hier gültig. Hier hat man es nun aber mit kontinuierlichen, mehrdimensionalen Wertebereichen der Variablen zu tun.

Es sei hier noch kurz bemerkt, daß die algorithmische Ebene oft auch Mikroprogrammierungsebene genannt wird. Die oft genannten Abstraktionsebenen "symbolisches Layout" und "metrisches Layout" sind weiter nichts als die Geometriesicht der Schalterebene bzw. der elektrischen Ebene.

## 3) Techniken der digitalen Simulation

Unter digitaler Simulation soll hier der Bereich zwischen Systemebene und Schalterebene verstanden werden, wobei die Schalterebene sowohl mit Methoden der digitalen [LER83, KAH84, LEW88] wie auch der analogen Simulation angegangen wird. Es ist nun Aufgabe der Simulationsalgorithmen, ein oder mehrere Modellierungskonzepte effizient auf einen Gastrechner abzubilden. Hier soll nicht der Fall betrachtet werden, daß die Architektur des Gastrechners dem Modellierungskonzept angepaßt ist (Prinzip der Hardwareakzelleratoren [ABR85, DEN82, HAF85, RSV88]) sondern von einem sequentiellen v. Neumann-Rechner ausgegangen werden. Bevor die verschiedenen Simulationstechniken diskutiert werden sollen, muß zunächst auf mögliche interne Modellierungskonzepte eingegangen werden:

Als einfaches und zugleich universelles Konzept bietet sich hier das Modell der "guarded commands" an. Dies wird gebildet durch eine (ungeordnete) Menge von Ausdrücken der Art:

$$\mathbf{guard}_i : \mathbf{action}_i$$

mit der Semantik, daß $\mathtt{action}_i$ immer dann ausgeführt wird, wenn $\mathbf{guard}_i$ wahr wird. Das Modell schließt nicht aus, daß mehrere Aktionen gleichzeitig ausgeführt werden oder daß noch aktive Aktionen neu initiert werden.
Um Modelle der Systemebene (objektorientierter Ansatz) auf dieses interne Konzept abzubilden, wird man das Eintreffen einer Nachricht als "guard" auffassen, die Durchführung der aufgerufenen Methode als "action". Für die algorithmische Ebene, wo es gilt, nebenläufige Algorithmen darzustellen, bietet es sich an, diese Algorithmen in interpretierte Petrinetze zu transformieren. Dann kann man die Eigenschaft einer Transition, aktiviert zu sein, als "guard" auffassen, das interpretierte Schalten als "action". Auf der Registertransferebene hat das Modell der "guarded commands" eine direkte Entsprechung, und auch stimulierte Gleichungssysteme, wie sie auf darunterliegenden Ebenen als externes Modellierungskonzept vorliegen, können in dieses interene Modell übersetzt werden, indem man entweder davon ausgeht, daß der "guard" stets wahr ist oder aber eine Entsprechung in eine Wertänderung eines beliebigen Arguments der "action" hat.
Dieses Modell der "guarded commands" wird nun durch den eigentlichen Simulationsalgorithmus auf die Zielmaschine abgebildet, wobei sich drei Grundtechniken herausgebildet haben:

- "streamline code simulation" (SCS) auch "compiled mode simulation" genannt,
- "equitemporal iteration" (EI),

- "critical event simulation" (CES).

Das SCS-Verfahren [ISH85, KOS85] beruht auf der Idee, unmittelbar ausführbarer Code des Gastrechners aus der Schaltungsbeschreibung zu generieren. Effizient ist dieses Verfahren, wenn das zu simulierende Objekt kombinatorisch oder zumindestens strikt sequentiell ist und kein Bedarf an Zeitinformation besteht. Es sei zunächst der kombinatorische Fall betrachtet. Hier kann die Schaltungsstruktur durch einen "dag" beschrieben werden, dessen Knoten bzgl. ihres längsten Pfades zu einen Primäreingang numeriert werden können ("levelizing"): **level** $(n_i) = o$ für alle Primäreingänge $n_i$ und **level** $(n_j) = (1+$ **max** $\{$**level** $(n_k) | \exists$ von $n_k$ zu $n_j)\}$ sonst. Nun kann ein Knoten auf einer Ebene `i` nur solche auf Ebene `j` mit $i <$ `j` beeinflussen. Ordnet man die Instruktionen zur Berechnung der Knoten in der Reihenfolge aufsteigender Ebenen an (mit beliebiger Reihenfolge innerhalb einer Ebene) so berechnet man mit minimalem Aufwand die Werte der Primärausgänge als Funktion der Werte an den Primäreingängen. Strikt getaktete sequentielle Schaltwerke re
duzieren sich, falls man an Zeitinformation nicht interessiert ist, auf eine Sequenz von Berechnungen der $\lambda$- und $\delta$- Funktionen (Ausgabe und Folgezustand), können also mit der selben Methodik mit erfaßt werden. Grundsätzlich kann man auch beliebige asynchrone Systeme, deren Zeitverhalten von Interesse ist, mit dem SCS-Verfahren simulieren, falls man Totzeitglieder durch Schieberegister nachbildet, doch führt dies zu ineffizienten Lösungen. Das Hauptanwendungsgebiet von SCS ist daher auch die Fehlersimulation, also ein Gebiet, in dem Informationen über das Zeitverhalten ohne Interesse ist.
Die Äquitemporale Iteration (EI) ist eine einfache tafelgetriebene Methode. Es wird angenommen, daß das zu simulierende System durch eine Menge von Komponenten $(c_i, a_i, d_i)$ beschrieben ist. Dabei bedeutet $c_i$ die Ausführbarkeitsbedingung der Komponente $i$, $a_i$ ihre Aktion und $d_i$ ihre Zieldatenobjekte. Alle diese Komponenten werden nun iterativ überstrichen. Ist die Ausführbarkeitsbedingung $c_i$ einer Komponente erfüllt, wird ihre Aktion $a_i$ ausgeführt. Während die Argumente einen globalen Speicher entnommen werden, werden die Ergebnisse in einen lokalen Puffer geschrieben. Diese Puffung der Ergebniswerte sichert die Unabhängikeit des Simulationsergebnisses von der Anordnung der Komponenten in einem Iterationsdurchlauf. Nach Abarbeiten aller Komponenten endet eine Iteration damit, daß die lokalen Puffer in den globalen Speicher kopiert werden und die simulierte Zeit um einen nicht notwendigerweise fixen, aber für alle Komponenten gleichen Betrag erhöht wird.
Der nachfolgende Algorithmus verdeutlicht diese Vorgehensweise:

```
begin
     simulated_time := start_time;
     final_time     := stop_time;
     while simulated_time <= final_time do
        begin
   for i:= 1 to component_number do
      if c(i) then d_buffer [i] := a (i);
           simulated_time := simulated_time + increment;
   for i:= 1 to component_number do
      do [i] := d_buffer [i]
        end
end
```

Die EI-Methode paßt ausgezeichnet zu einer Einheitsverzögerungsannahme und damit auch zu taktorientierten Zeitmodellen, wie sie auf der Registertransferebene häufig sind. Durch ähnliche Methoden, wie im SCS-Ansatz, ist auch hier eine Behandlung von realen Zeitmodellen möglich. Doch tendiert auch die EI-Methode dann zur Ineffizienz. Da zu jeden simulierten Zeitpunkt jede Komponente besucht wird, ist der EI-Ansatz nur dann effizient, wenn die Wahrscheinlichkeit hoch ist, daß eine bestimmte Komponente zu einen bestimmten Zeitpunkt instabil ist. Diese Wahrscheinlichkeit sinkt natürlich bei feinerer Auflösung der Zeitachse oder bei gröberer Auflösung der Werteachse. Die EI-Methode wird daher hauptsächlich dann eingesetzt, wenn der Wertebereich sehr groß (Analogsimulation) oder die zeitliche Auflösung relativ grob ist (zyklenorientierte RT-Simulation).

Die Simulationsmethode die sich auf Berechnungen beschränkt, die nicht die Identität bedeuten, ist die "critical event"-Simulation (CES [JAE83, SZY72]). Hier wird als einzige Voraussetzung an das zu simulierende Objekt "Determiniertheit" gefordert:

(i) Der Zeitpunkt des nächsten Auftretens eines bekannten Ereignisses ist vorhersagbar.

(ii) Ist der Zeitpunkt des nächsten Auftretens eines bestimmten Ereignisses nicht vorhersagbar, so findet dieses Ereignis nicht statt, bevor es nicht durch das Stattfinden anderer Ereignisse vorhersagbar geworden ist.

Weiterhin nimmt der CES-Ansatz an, daß das zu simulierende System in Komponenten zerteilt ist und die Abhängigkeitsbeziehungen zwischen diesen Komponenten statisch und bekannt sind. Ist Komponente B direkt abhängig von Komponente B, so wird A Beeinflusser von B und B Beeinflußter von A genannt. Mit dieser Informatik kann der CES-Algorithmus präzise bestimmen, welche Komponenten durch die Berechnung einer Komponente beeinflußt werden und zusammen mit der Zeitinformation kann auch vorhergesagt werden, wann dies geschieht. Diese Information muß nun lediglich in eine nach der simulierten Zeit sortierten Warteschlange gehalten werden. Der Algorithmus entnimmt dieser Warteschlange den obersten Wert, k setzt seine simulierte Zeit auf den dort gespeicherten Wert, führt die aufgeführte Operation durch und sortiert die Ereignisse für die beeinflußten Komponenten nach den Sortierkriterium "simulierte Zeit" diese zukünftigen Ereignisse in die Warteschlange ein.
Das folgende Skelett eines Algorithmus mag dies verdeutlichen:

```
type  event  = record
  component_id, event_time : integer
  new_value                : word
               end

var  current_event, new_event : event;
     event_queue : queue of event;
     current_time, queue_fill : integer;
     changed                  : boolean;

begin
     time := 0; final_time := stop_time;
     while time <= final_time & queue_fill <> 0 do
       begin
  current_event := remove (event_queue);
  current_time  := current_event.time;
  changed := data [current_event.component_id] <>
     current_event, new_value;
          data [current_event.concurrent_id] := current_event.new_value;
  if changed then begin
for i:= 1 to influencee_card [current_event.component_id] do
  begin
    component := influencee [current_event.component.id,i]
    if executable (component) then begin
new_event.component_id := component;
new_event.new_value    := action (component);
new_event.event_time   := current_time +
  elapses (component);
                        insert (eventqueue, new_event)
                    end
                end
          end
```

```
            queue_fill := test (event_queue)
        end
end
```

Die Ereignisschlange kann entweder durch Sortieren verwaltet werden oder nach dem sogenannten "time wheel"-Verfahren. Hierbei geht man von einem "Zeitrad" mit n "Schlitzen" aus, das die n nächsten Zeitpunkte beeinhaltet. Spätere Zeitpunkte können modulo n unter Angabe noch nötiger "Umdrehungen" des Rades den gleichen "Schlitzen" zugeordnet werden. Bei dieser Alternative spart man sich zwar den Sortieraufwand (er wird durch ganzahlige Division durch n und Modulobildung modulo n ersetzt, d.h. durch Teilststringbildung, wenn n eine Zweierpotenz ist), muß aber nun in einer gewählten Auflösung alle Zeitpunkte besuchen. Dies ist zwar weniger als im EI-Ansatz, wo man zu jedem Zeitpunkt jede Komponente besuchen muß, doch lohnt sich bei relativ kurzen Ereignisschlangen dieser Aufwand im Vergleich zu dem dann geringen Sortierwaufwand selten.

Der CES-Ansatz ist die heute gebräuchliche Simulationsmethode der digitalen Simulation. Er ist hochgradig effizient und wird darin nur von dem in seiner Anwendbarkeit sehr beschränkten SCS-Ansatz übertroffen. Zudem erlaubt er ohne merkliche Leistungseinbußen fast beliebig komplexe Zeitmodelle. Beispiele für derartige Simulatoren aus der Gatterebene sind HILO, CADAT und DISIM.

# 4) Mehrebenensimulation

Der traditionelle Ansatz, für jede Abstraktionsebene einen dedizierten Simulator anzubieten, führt bei immer komplexeren Systemen zu nicht unerheblichen Problemen. Das erste Problem ist, daß während des Entwurfsprozesses verschiedene Simulatoren zu benutzen sind. Somit muß der Benutzer mit einer Reihe recht komplexer Softwaresysteme vertraut sein und die Entwurfsdaten müssen mehrfach transformiert werden.
Der wesentliche Nachteil jedoch ist, daß stets das gesamte Entwurfsobjekt auf einer Abstraktionsebene modelliert und simuliert werden muß. Der typische Entwurfsstil der stückweisen Verfeinerung wird dadurch überhaupt nicht unterstützt. Für einen derartigen Entwurfsstil muß es möglich sein, ein Modul in einem Simulationsmodell durch eine andere Beschreibung desselben Moduls auf einer niedrigeren Abstraktionsebene zu ersetzen und das gesamte System zu simulieren, d.h. das Modul auf niedrigerer Ebene, das gerade von Interesse ist, zusammen mit seiner auf höheren Ebenen formulierten Umgebung. Daher rührt die Forderung nach Vielebenensimulatoren. Dabei versteht man unter "Multi Level"-Simulationssystem, das mehr als eine Abstraktionsebene überdeckt. Derartige Systeme unterstützen nicht notwendigerweise Beschreibungen, die gleichzeitig mehrere Ebenen überdecken. Falls auch solche Beschreibungen unterstützt werden, spricht man von "Mixed Level"-Simulation. Es gibt zwei Hauptansätze für die "Multi Level/Mixed Level"-Simulation:
Entweder wird ein Satz von dedizierten Simulatoren zu einem mehr oder weniger integrierten System gekoppelt (Multisimulatoransatz), oder ein Breitbandsimulator wird angeboten.

## 4.1 Multisimulatoransatz

Ein Ansatz einer Mehrebenensimulation [MEB85] ist die Kopplung existierender dedizierter Simulatoren. In diesem Fall sind drei Hauptprobleme zu lösen:

(i) Der Datenaustausch zwischen den beteiligten Simulatoren,

(ii) die Synchronisation der Aktivitäten der verschiedenen Simulatoren,

(iii) Eine Benutzerschnittstelle, die eine uniforme Kommunikation mit dem gesamten Simulationssystem erlaubt.

Hier soll nur auf die ersten beiden Aspekte eingegangen werden.

### Datenaustausch

Durch das Identifizieren von Primäreingängen eines auf einen Simulator laufenden Modells mit Primärausgängen eines auf einen anderen Simulator laufenden Modells werden Verbindungskanäle festgelegt, auf denen ein Datenaustausch stattfindet. Die einfachste Situation ist im Fall identischer Wertebereiche und identischer Datendarstellungen gegeben. Diese Situation tritt im gegebenen Kontext jedoch selten auf, da typischerweise auf verschiedenen Abstraktionsebenen verschiedene Wertebereiche für die Variablen benutzt werden und außerdem unterschiedliche Simulatoren oft auch unterschiedliche Datendarstellungen benutzen. Falls sich nur die Darstellung unterscheidet, muß bei jedem Datenaustausch eine einfache Konversion vorgenommen werden.
In den meisten Fällen jedoch liegen verschiedene Wertebereiche vor. Typischerweise werden auf niedrigeren Abstraktionsebenen Wertebereiche höherer Kardinalität benutzt. Gewisse Teilmengen dieser Wertebereiche werden dann auf höheren Ebenen als gewisse Werte interpretiert. Die Situation ist relativ einfach, falls es eine Partition auf dem Wertebereich der niedrigeren Ebene derart gibt, daß die Repräsentanten der verschiedenen Klassen gerade den Wertebereich auf der höheren Ebene bilden.
Es ist also relativ einfach, Daten von einem Simulator auf niedrigem Abstraktionsniveau zu einem auf hohem zu übergeben. In den meisten Fällen kann die Partitionierung durch eine einfache Schwellwertfunktion definiert werden. Die umgekehrte Richtung ist erheblich schwieriger. Hier wird vom Sender nur die Information geliefert, in welche Klasse möglicher Werte auf der niedrigeren Ebene der gelieferte Wert gehört. Die Identifikation des vom Simulator auf niedriger Ebenen zu selektrierenden individuellen Wertes muß entweder global oder individuell für jede solche Verbindungsleitung angegeben werden.

### Beispiel:

Man betrachte die beiden im obigen Beispiel benutzten Wertebereiche, **seven** um auf Schalterebene, **three** um auf Gatterebene benutzt zu werden. Es sei angenommen, daß **connect** eine unidirektionale Leitung von einem Modell (Simulator) auf Gatterebene zu einem auf Schalterebene ist. Ein allgemeine Annahme wie "Es werden nur Gatter mit Ausgangstreibern benutzt" resultiert in einer Abbildung
**three** → **seven** mit **1** ∈ **three** → **1** ∈ **seven**, **0** ∈ **three** → **0** ∈ **seven**, **X** ∈ **three** → **X** ∈ **seven**.
Eine individuelle Annahme über diese Leitung wie "connect ist der Ausgang einer Speicherzelle ohne Lesepuffer" mag die folgende Abbildung **three** → **seven** implizieren:

**1** ∈ **three** → **H** ∈ **seven**,
**0** ∈ **three** → **L** ∈ **seven**,
**X** ∈ **three** → **Y** ∈ **seven**.

Die Situation ist besonders kompliziert, falls diskrete Wertebereiche in kontinuierliche, wie sie bei der elektrischen Simulation benutzt werden, abgebildet werden müssen. In diesem Fall muß ein Modell des Treibers, dessen Existenz angenommen werden kann (obwohl er auf der höheren Ebene nicht explizit genannt wird), durch das Kopplungssystem angegeben werden. Hier scheint es sinnvoller zu sein, für die verschiedenen vorkommenden Leitungen individuelle Abbildungen
vorzusehen.

### Synchronisation

Die verschiedenen Simulatoren eines Multisimulatorsystems zusammen modellieren das Gesamtsystem. Daher müssen sie synchron gehalten werden. Allerdings ist es nicht notwendig, sie vollständig synchron zu halten, sodaß zu jeder Zeit jeder Simulator den Zustand seines Teils zu genau demselben simulierten Zeitpunkt darstellt. Doch müssen die Simulatoren mindestens so weit synchronisiert werden, daß jeder Datenaustausch zur korrekten Zeit für den Sender wie auch den Empfänger stattfindet. Es gibt drei Hauptansätze, die beteiligten Simulatoren zu synchronisieren.

Im Supervisoransatz sichert ein übergeordneter zentraler Supervisor eine jederzeit vorhandene Synchronität der beteiligten Simulatoren. Ebenfalls synchron laufen die Simulatoren im "time messager"-Verfahren [CHM81, PMW80], doch wird hier der Supervisor durch einen verteilten Ansatz ersetzt. Die "time warp"-Methode [JES82] schließlich geht von einem asynchronen Betrieb der Simulatoren aus. Alle Ansätze resultieren in einer geringfügigen Modifikation der beteiligten Simulatoren. Im Fall eines zentralisierten Supervisors muß jeder Simulator die Kontrolle an den Supervisor abgeben, bevor er eine Zeitfortschaltung vornimmt. Der Supervisor identifiziert nun den Simulator, der die am wenigsten entfernte Zukunft behandeln will, und übergibt die Kontrolle an diesen Simulator. Event Scheduling scheint durch seine Flexibilität die am besten geeignete Methode für den Supervisor zu sein. Die einzelnen Simulatoren jedoch sind nicht auf diesen Typ eingeschränkt. Es wird nur gefordert, daß der, wie immer auch geartete Hauptzyklus für eine Kommunikation mit dem Supervisor aufgeschnitten
werden kann. Der geeignete Aufschneidepunkt liegt im Falle von Event Scheduling unmittelbar hinter dem Ort, an dem das nächste Ereignis aus der Ereignisschlange extrahiert wurde. In den Fällen SCS und EI sollte der Zyklus unmittelbar vor Start eines neuen Zyklus aufgeschnitten werden. Der Supervisor-Ansatz "übersynchronisiert" die Simulatoren, da kein gegenseitiges Überholen der Simulatoren möglich ist, selbst dann nicht, wenn in der Überlappungsperiode keine Kommunikation stattfinden würde. Ein weiterer Nachteil ist, daß stets nur ein Simulator zu einem Zeitpunkt aktiv sein kann. Somit ist diese Methode für Multiprozessorsysteme weniger geeignet.

Für Multiprozessorsysteme ist der "time messager"-Ansatz geeignet. Hier wird die Synchronisation auf die beteiligten Simulatoren verteilt. Bevor ein Simulator auf einen neuen Zeitpunkt fortschaltet, teilt er dies allen anderen Simulatoren mit, mit denen er Daten austauscht. Erst wenn diese ihm eine Freigabe geben ( da sie ihm bis zu diesen neuen Zeitpunkt keine Daten schicken werden) findet die Zeitfortschaltung und der damit verbundene Zustandswechsel tatsächlich statt. Ist der Supervisor-Ansatz das übersynchronisierende Extrem, so ist der "Time-warp"-Ansatz das untersynchronisierende. In diesem Fall werden vollständig frei laufende Simulatoren ohne jegliche Synchronisation angenommen. Allerdings muß nun ein Simulator, der eine Nachricht bezüglich eines Zeitpunkts erhält, der vor seinem aktuellen liegt, auf diesen früheren Zeitpunkt zurückgesetzt werden.
Es sei angenommen, es existieren die Simulatoren $S_1$, derzeit an dem lokalen simulierten Zeitpunkt (LST) $ts_1$ und $S_2$ an LST $ts_2$. Es sei weiter angenommen, daß $S_1$ eine Nachricht m = (data,$t_m$) an $S_2$ sendet, wobei $t_m$ den simulierten Zeitpunkt der Nachricht angibt. In jedem Fall wird gelten: $t_m \geq ts_1$. Es gibt nun zwei Möglichkeiten:

$(i)$ $\quad t_m > ts_2$,
$(ii)$ $\quad t_m \leq ts_2$.

Im Fall `(i)` wird `m` wie ein normaler Eingabestimulus behandelt. Im Fall `(ii)` ist der Fall komplizierter. Wir müssen nun annehmen, daß $S_2$ in einen Zustand vollständig vor $t_m$ restauriert (zurückgerollt) werden kann. In der ersten Phase der Behandlung von Fall `(ii)` genannt "Rückroll-Phase", wird der am nächsten liegende gespeicherte Zustand von $S_2$ zu LST $t_r < t_m$ geladen. Nun kann jedoch $S_2$ im Zeitraum zwischen $t_r$ und $ts_2$ Nachrichten zu anderen Simulatoren gesandt haben. Diese Nachrichten sind nun aber illegal. In der zweiten Phase, genannt "Rücknahme-Phase", werden all diese Nachrichten zurückgenommen, einfach dadurch, daß man sie nochmals sendet, jedoch mit einem "Rücknahme-Marker" versehen. Der Erhalt einer solchen Nachricht bewirkt beim empfangenden Simulator, daß die entsprechende Nachricht aus der Nachrichtenschlange gelöscht wird und daß ansonsten auf diese Nachricht wie auf eine normale Nachricht reagiert wird. D.h., falls seine LST bereits den Zeitpunkt der zurücknehmenden Nachricht passiert hat, muß er Fall `(ii)` ebenfalls durchführen. Dadurch kann sich der Rücknahmeprozeß für eine gewisse Zeit durch das gesamte Multiprozessorsystem fortsetzen. Weil aber durch den Rücknahmeprozeß Einträge in Schlangen endlicher Länge gelöscht werden und keine neuen Einträge entstehen, ist der Prozeß endlich. In der letzten Phase, genannt "Vorrück-Phase", muß $S_2$ bis zum LST $t_m$ simulieren.

Die "Time-warp"-Methode ist sehr allgemein. Sie macht keinerlei Annahmen über die zu simulierenden Systeme über solche hinaus, die für jede Simulation notwendig sind. Alle Typen von Simulationsalgorithmen können so modifiziert werden, daß sie in einer "time-warp"-Umgebung funktionieren. Die involvierten Simulatoren können nebenläufig laufen, sodaß die Methode für Multiprozessor-Umgebungen ideal geeignet ist. Der Nachteil der Methode rührt von der zu optimistischen Annahme her, daß die beteiligten Simulatoren "ungestört" laufen können. Im Falle einer Störung durch eine in der lokalen Vergangenheit liegende Nachricht entstehen jedoch enorme Kosten. Untersuchungen [ ] deuten darauf hin, daß die "time warp"-Methode nur bedingt einsetzbar ist.

### 4.2. Breitbandsimulation

Hat man eine Breitbandsprache wie DACAPO [DAC87] oder VHDL [VHD87], so scheint der natürlichste Ansatz ein Breitbandsimulator zu sein, um eine derartige Sprache unmittelbar zu unterstützen. Dieser Ansatz hat eine Reihe von Vorteilen. Zunächst wird
die gesamte Simulation durch eine einzige Umgebung durchgeführt. Das zu simulierende Objekt kann, wie es beschrieben ist, behandelt werden, ohne es zu transformieren oder auf verschiedene Simulatoren zu verteilen. Es werden auch keine trickreichen Beschreibungen notwendig, um Restriktionen bestimmter Simulatoren zu umschiffen. Stückweise Verfeinerung ist hier auch kein Problem, da man stets innerhalb einer Umgebung bleibt.
Natürlich gibt es auch bei diesem Ansatz Nachteile. Zunächst ist möglicherweise die überdeckte Bandbreite sehr groß. Somit muß ein recht komplizierter Simulationsalgorithmus, der eine Reihe von Modellierungskonzepten in einer heterogenen Menge von Wertebereichen unterstützt, implementiert werden. Derartige allgemeine Algorithmen tendieren dazu, weniger effizient zu sein, als dedizierte. Dies liegt daran, daß dedizierte Algorithmen bekannte Restriktionen des Bereichs, den sie abdecken müssen, ausnutzen können, während Breitbandalgorithmen mit einer Reihe möglicher Situationen rechnen müssen. Im Fall von DACAPO III wurde dieses allgemeine Problem dadurch gelöst, daß so viel wie möglich in unmittelbar ausführbaren Code zu übersetzt wird. Nur der elementarste Scheduling-Algorithmus existiert als hochoptimierter Interpretationsalgorithmus. Die restlichen Teile werden für jedes zu simulierende Modell individuell in ausführbaren Code des Gastrechners übersetzt. Somit wird im Vergleich zu dedizierten Simulatoren nahezu kein Overhead produziert. Dieser Ansatz wurde durch die modernen Compilergeneriermethoden möglich, die es erlauben, automatisch hochoptimierende Codegeneratoren relativ einfach und portabel zu generieren.
Daß bei der heutigen Technologie Breitbandsimulatoren den zu bevorzugenden Ansatz darstellen, bedeutet nicht, daß Multisimulatorsysteme überhaupt keinen Sinn mehr haben. Es gibt sehr wohl Situationen, in denen diese Methode vorzuziehen ist. Eine typische solche Situation ist gegeben, falls ein Halbleiterhersteller nur Simulationsergebnisse eines bestimmten Herstellers akzeptiert. Eine ähnliche Situation ist gegeben, falls es Module von Komponenten, die in einem Entwurf benutzt werden sollen, nur in einer bestimmten Sprache gibt. Schließlich scheint dieser Ansatz am besten geeignet zu sein, um gemischte Digital/Analog-Simulatoren zu bauen.

## 5) Literatur

**[ABR] M. Abramovici et al. :**
A Logic Simulation Machine
IEEE T o CAD of Inctegrated Circuits and Systems, Vol. CAD-2, No. 2

**[CHM] K.M. Chandy, J. Misra :**
Asyncronous Distributed Simulation via a Sequence of Parallel Computations
Comm. ACM 24, 11 Apr. 1981

**[DAC87] - :**

DACAPO III System-User-Manual
DOSIS GmbH, Dortmund,

**[DEN82] M.M. Dennau :**
The Yorktown Simulation Engine: Architecture and Hardware Description
in: Proceedings of 19th DAC, 1982

**[FIS78] G.S. Fishman :**
Principles of Discrete Event Simulation
John Wiley & Sons, 1978

**[GAJ87] D.D. Gajski :**
The Structure of a Silicon Compiler
in: Proceedings of IEEE ICCD, 272-276, 1987

**[HAF85] W. Hahn, K. Fischer :**
High Performance Computing for Digital Design Simulation
in: Proceedings IFIP VLSI'85, North Holland, 1985

**[ISH85] N. Ishiura et al. :**
High-Speed Logic Simulation Using a Vector Processor
in: Proceedings IFIP VLSI'85, North Holland, 1985

**[JAE83] U. Jaeger :**
Logik- und Fehlersimulation in dem Programmsystem DISIM
in: Seminarunterlagen Praxis der Grossintegration,
FB Elektrotechnik, Univ. Dortmund, 1983

**[JES82] D. Jefferson, H. Sowizral :**
Fast Concurrent Simulation Using the Time Warp Mechanism,
Part I: Local Control
Rand Corporation, Rand Note N-1906-AF, 1982

**[KAH87] M. Kawai, J.P. Hayes :**
An Experimental MOS Fault Simulation Program CSASIM
in: Proceedings of 21th DAC, 1984

**[KOS85] S. Koepper, C. Starke :**
Logiksimulation komplexer Schaltungen für sehr große Testlängen
in: NTG Fachberichte, Band 87, 1985

**[LEW88] K.D. Lewke :**
Ein Modell zur ereignisgesteuerten Simulation von MOS-Transistornetzwerken auf Schalterebene
Dissertation Univ.-GH-Paderborn, FB 17, 1988

**[LER83] K.D. Lewke, F.J. Rammig :**
Description and Simulation of MOS Devices in Registertransfer Languages
in: Proceedings IFIP VLSI'83, North Holland, 1983

**[MER85] J. Mermet :**
The CASCADE Hierachical Multilevel Mixed Mode (HM3) Simulator
in: Proceedings EUROMICRO'85, 1985

**[PMW80] J.K. Paecock, E.G. Manning, J.W. Wong :**
Syncronisation of Distributed Simulation Using Broadcast Algorithms
Computer Networks 4, 1, Feb. 1980

**[RAM87] F.J. Rammig :**
Multilevel Simulation Techniques
in: Proceedings COMPEURO'87, 1987

**[RAM89 F.J. Rammig :**
Systematischer Entwurf digitaler Systeme
Teubener, 1989

**[RSV88] F.J. Rammig, M. Schrewe, G. Vorloeper :**
A Transputer-Based Accelerator for Multilevel Digital Simulation
in: Proceedings EUROMICRO'88, North Holland, 1988

**[SZY72] S.A. Szygenda :**
TEGAS-2 Anatomy of genenal purpose test generation and simulation systems for digital logic
in: Proceedings ACM Design Automation Workshop, 1972

**[VHD87]-:**
IEEE Standard VHDL Language Reference Manual,
IEEE, iStd 1076 - 1987

**[ZEI76] B.Zeigler:**
Theory of Modelling and Simulation
John Wiley & Sons, 1976