Analytische Transaktionsmodellierung

Frank Slomka, Frank Bodmann, Karsten Albers, Institut für Eingebettete Systeme/Echtzeitsysteme,
Fakultät für Ingenieurwissenschaften und Informatik, Universität Ulm

Kurzfassung

Der Beitrag führt eine neue Methodik zur analytischen Beschreibung von Transaktionen in eingebetteten Echtzeitsystemen ein. Dabei wird besonderer Wert auf eine automatisch aus der algoritmischen Ebene ableitbare Ereignismodellierung auf der Systemebene gelegt. Durch die Einführung eines erweiterten Ereignisstrommodells gelingt es die Ergebnisse aus der Analyse der Systemalgorithmen mit der klassischen Echtzeittheorie zur analytischen Verifikation von Echtzeitsystemen zusammenzuführen. Mit den aus dieser Methodik resultierenden Ereignisabhängigkeitsgraphen lassen sich dann verteilte Echtzeitsysteme elegant modellieren und analysieren.

1.Einleitung

Im Entwurf integrierter Schaltungen hat sich die Aufteilung des Entwurfsproblems auf unterschiedliche Abstraktionsebenen bewährt. Dabei sind die Modelle der klassischen Entwurfsebenen, Layout-, Transistor- oder Schaltkreis- und Register-Transfer Ebene mathematisch verknüpft. Je nachdem wie detailliert man die Modelle ausführt, kann man z.B. mittels Parameterextraktion aus dem Layout die elektrischen Eigenschaften der Transistorschaltung für eine Netzwerksimulation mit dem Schaltungssimulationsprogramm *Spice* extrahieren; oder man ermittelt auf der Netzwerkebene die Verzögerungszeiten der steigenden und fallenden Flanken der Logikgatter. Diese Parameter können dann bei der Verifikation auf der entsprechenden Ebene verwendet werden. Reicht diese Verifikationsmethodik für digitale Systeme bis zur Register-Transfer Ebene und den Instruktionssätzen der Prozessoren ist die Methodik hinauf zur Systemebene noch lückenhaft. Zwar existieren viele Modelle digitale Systeme auf der Systemebene zu beschreiben, zu modellieren und zu simulieren, bzw. zu analysieren, die Methoden Parameter aus der algorithmischen auf die Systemebene zu transferieren beschrängt sich jedoch auf die Ableitung von Taskausführungszeiten. Kommen für den Entwurf signalverarbeitender Systeme, als typische Vertreter der Klasse der datenflußdominanten Systeme, häufig Synchrone Datenflußgraphen (SDF) zum Einsatz, beschreibt man klassische, Ereignissysteme häufig mit Hilfe eines einfachen periodischen Ereignis- und Tasksmodells. Auf der Grundlage dieses Modells wurde dann die klassische Echtzeitanalyse und Schedulingtheorie aufgebaut. Die Grundelemente dieser Theorie sind zunächst einmal unabhängige, und in späteren Erweiterungen, kommunizierende Prozesse. Ähnlich wie beim Übergang von der Transistor- auf die Logikebene die Verzögerungszeiten der Gatter bestimmt werden, kann man aus dem Programmcode der einzelnen kommunizierenden Prozesse die Ausführungszeiten dieser auf vorgegebenen Prozessoren oder Hardwarekomponenten ermitteln. In den letzten Jahren spielt aber zunehmend die Kommunikation der Prozesse eine Rolle. Da die klassische Echtzeittheorie von einem vereinfachten Modell, Kommunikation findet immer am Ende der Programmausführungszeit statt, aus. Dies führt dazu, dass es nur wenige Erweiterungen der Scheduling-analyse für verteilte Systeme gibt., z.B. [2], [3], [5]. Allerdings ist allen diesen Erweiterungen gemeinsam, dass der Einfluß des Programms eines Prozesses auf die Kommunikationsrate zwischen den Prozessen unberücsichtigt bleibt. Programme, die in einem parallelen, verteilten System zu beliebigen Zeiten Kommunikationsereignisse erzeugen werden heutzutage meist simulativ verifiziert. Dieser Beitrag beschreibt eine Methodik zur Berechnung der Kommunikationsereignisse verteilter Systeme aus dem Programmcode der kommunizerenden Prozesse und eine analytische Performanzanalyse auf der Systemebene, die diese Informationen nutzt. Grundlage des Verfahrens sind die von uns neu eingeführten Ereignisabhängigkeitsgraphen [1]. Im wesentlichen handelt es sich dabei um Task- oder besser Prozessgaphen in denen Knoten die einzelnen Prozesse des Systems und die Kanten die Kommunikationsbeiehungen modellieren. Im Gegensatz zum klassischen Datenflußgraphen werden in den Ereignisabhängigkeitsgraphen auch die Ereignisraten der Kanten beschrieben. Um ein möglichst allgemein-gültiges Modell zu erhalten, verwenden wir dazu das Ereignismodell der

** The research described is partly supported by the Deutsche Forschungsgemeinschaft under grant SL 47/1-1, SL 47/2-1*

Ereignisströme [4]. Im folgenden führen wir kurz in die Grundlagen unseres neuen Modells ein, dann führen wir in die Ereignisabhängigkeitsanalyse ein. Ausgehend von einfachen Beispielen, zeigen wir die Schwächen des Modells zur Transaktionsanalyse auf und führen daraufhin eine Erweiterung des Ereignismodells ein. Diese einfache Erweiterung der Ereignisströme erleicht die Modellierung und reduziert die Komplexität der Analysealgorithmen. Zum Schluß werden die Ergebnisse einer ersten Designstudie präsentiert.

2.Systemmodell

Um eine Systemarchitektur analysieren und verifizieren zu können wird ein Modell benötigt. Auf der Systemebene besteht ein Entwurf aus unterschiedlichen Komponenten: Zum einen beschreibt man die Hardware in Form von zusammengeschalteten Prozessoren, Bussen und Leitungen und unterschiedlichen Speicherbausteinen. Prozessoren können in dieser Vorstellung sowohl programmierbar sein, also Software ausführen, aber auch als festverdrahtete Spezialprozessoren für bestimmte Anwendungen optimiert sein. Die Software des Systems kann auf der anderen Seite wie folgt strukturiert werden: Auf einem Prozessor ermöglicht ein Betriebssystem die quasiparallele Ausführung mehrerer Anwendungsprogramme und stellt Dienste bereit, damit diese Prozesse miteinander kommunizieren können. Ein Prozeß ist dabei definiert durch ein Anwendungsprogramm, den zu einer Ausführung des Programms gehörenden Daten und den jeweiligen Prozessorkontext (Registerinhalt). Die Prozesse werden i.A. nacheinander ausgeführt, können aber auch durch Prozesse höherer Priorität unterbrochen werden. Die Ausführungsreihenfolge der Prozesse wird durch einen Ablaufplan dynamisch festgelegt. Im allgemeinen läßt sich ein verteiltes eingebettetes System zur Analyse der Rechenlastanforderungen als eine Menge kommunizierender Programme oder Prozesse beschreiben.

2.1. Flußgraphen

Um die Eigenschaften eines Programms analysieren zu können, z.B. die Anzahl der benötigten Registerwechsel auf einerm vorgegebenen Prozessor, werden im klassischen Compilerbau die syntaktischen Struktur des Programmquelltextes in einen Flußgraphen übersetzt. Häufig wird ein derartiger Graph auch Basisblock- oder Kontrollflußgraph genannt. Der spätere Übersetzungsprozeß des Compilers aber auch ggf. durchzuführende Optimierungen des Codes erfolgen auf der Grundlage des Flußgraphen. Ein solcher Programmflußgraph besteht aus Knoten und Kanten. Die Kanten repräsentieren dabei jeweils einen Basisblock des Programms. Unter einem Basisblock versteht man mehre Befehlszeilen des Programms, wobei gilt, daß der Kontrollfluß des Programms in einem Basisblock nicht verzweigt. Daher beschreibt man den Kontrollfluß des Programms mit Hilfe der gerichteten Kanten des Basisblockgraphen. Dabei wird der Programmfortschritt durch einfache gerichtete Kanten zwischen den Knoten des Basisblockgraphen beschrieben. Im Falle von Verzweigungen hat ein Knoten mehrere Nachfolgeknoten und die Verzweigungsbedingung wird an den Kanten des Graphen notiert.

Um die im folgenden Aufsatz entwickelte Ereignisabhängigkeitsanalyse beschreiben zu können muß zwischen zwei unterschiedlichen Knotentypen des Basisblockgraphen unterschieden werden. Zum einen gibt es weiterhin die klassischen Knoten des Graphen, die den Steuer- und Grundblöcken des Programms entsprechen, zum anderen werden jetzt aber auch Knoten benötigt, die eine Sendeoperation zu einem anderen Prozeß beschreiben. Dies Knoten werden extra ausgezeichnet.

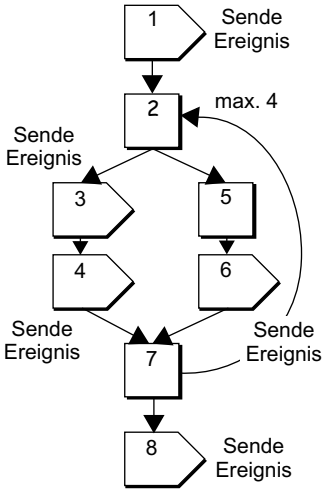

Bild 1: Erweiterter Flußgraph

2.2. Taskgraphen

Im Allgemeinen läßt sich jede beliebige Softwarestruktur mit Hilfe eines Taskgraphen beschreiben. Dabei stellen die Kanten des Graphen eine Aufgabe oder Funktion dar, z.B. eine komplexere Berechnungs- oder Kodierungsaufgabe, und die gerichteten Kanten beschreiben den Datenfluß zwischen den einzelnen Aufgaben. In der Literatur wird das Konzept des Taskgraphen durchaus unterschiedlich verwendet, meist wird jedoch das Modell eines einfachen Datenflußgraphen verwendet. Ist ein Knoten mit der Bearbeitung der Aufgabe fertigt beginnen die Nachfolgeknoten mit der Bearbeitung ihrer Aufgabe. Diese Knotenausführungen können Zeit verbrauchen und bei ausreichenden Hardwareressourcen auch parallel ausgeführt werden. Naturgemäß ist die Semantik dieses Modells begrenzt, da man nichts über die Signalraten auf den Kanten des Datenflußgraphen aussagen kann. Aus diesem Grund wurden die Synchronen Datenflußgraphen eingeführt. Dabei lassen sich unterschiedliche Aktivierungsraten an den Pfeil und die Basis der gerichteten Kanten des Graphen schreiben und damit Analysen durchführen. Da in diesem Modell jedoch keine Informationen bezüglich des Zeitverhaltens der Taskaktivierungen vorliegen, wird von konstanten Raten ausgegangen. Damit ist es möglich statische Ablaufpläne festzulegen und dann im späteren System zu

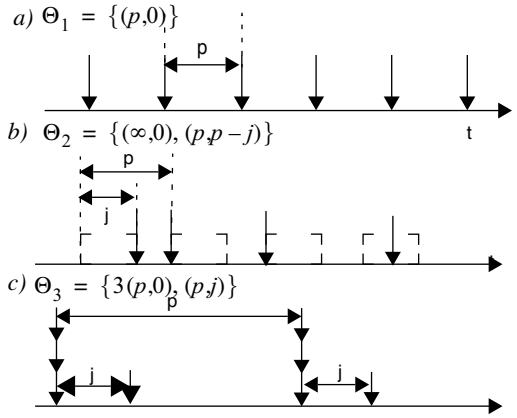

Bild 2: Modellierung von Ereignisschüben [4]

implementieren. Die Auswirkungen von dynamischen Ablaufplanungsverfahren, wie sie in komplexeren Systemen häufig eingesetzt werden, auf die Anwendung können damit nicht untersucht werden. Aus diesem Grund wollen wir die Taskgraphen erweitern. Zu diesem Zweck führen wir im folgenden das Ereignismodell der Ereignisströme ein.

2.3. Ereignisströme

Für die Analyse komplexer Echtzeitsysteme wurde von Gresser [4] das Modell der Ereignisströme entwickelt. Dabei ist er von der Vorstellung ausgegangen, daß man für den Nachweis, daß ein gegebenes Echtzeitsysteme alle vorgegebenen Zeitschranken einhält, den schlimmsten möglichen Fall des Auftretens von Ereignissen beschreiben muß. D.h. für eine Echtzeitanalyse ist es wichtig die kleinstmöglichen Ereignisabstände zu beschreiben. Darüber hinausgehende Informationen werden dann nicht mehr benötigt.

2.3.1 Das klassische Modell

Im klassischen Ereignisstrommodell von Gresser [4] werden mit Hilfe einer Menge von Ereignistupeln

$$ET = \binom{p}{a} \tag{1}$$

maximale Ereignisdichten beschrieben:

$$\Theta = \{ET_i\} = \left\{\binom{p_i}{a_i}\right\} \tag{2}$$

Dabei ist a ein Zeitintervall und p eine Wiederholperiode. Wichtig für die Interpretation eines Ereignisstroms ist nun die Stellung der Tupel $i \in N$ in der Menge θ. Das erste Tupel mit $i=1$ bezeichnet das minimale zeitliche Intervall in dem ein Ereignis auftreten kann. Daher ist a_1 immer Null. Die Ereignisse können sich mit der Periode pi wiederholen. Ist $p = \infty$ tritt ein Ereignis nur einmal auf (Bild 4a). Ein Ereignis, dass sich nicht periodisch wiederholt kann mit $p_1 = \infty$ beschrieben werden. Analog beschreibt a_2 das minimale Zeitintervall indem zwei

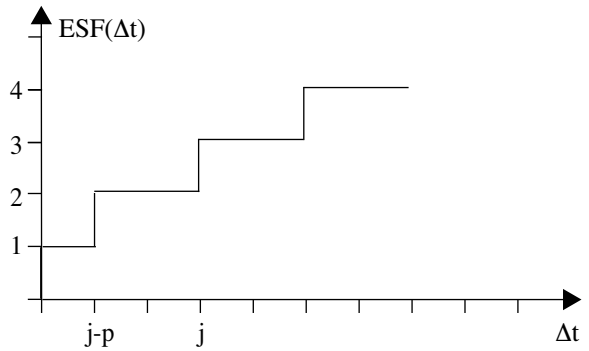

Bild 3: Ereignisdichtefunktion für Beispiel b) [4]

Ereignisse auftreten können (Bild 4b): Angenommen Ereignisse, die im System eintreffen schwanken (jittern) um eine gemeinsame Periode, und zwar in einem Intervall der Dauer *j*. Die minimale Dauer nach der zwei Ereignisse auftreten können, ist wenn ein Ereignis genau am Ende des Jitterintervalls auftritt und das sofort folgende Ereignis am Anfang. In diesem Fall ist der minimale zeitliche Abstand zwischen zwei Ereignissen $a_2 = p\text{-}j$. Das bedeutet auch, dass die Ereignisströme minimale Abstände beschreiben, also Ereignisse zwar nicht früher als in der Beschreibung, aber in jedem Fall später auftreten dürfen. Mit Ereignisströmen kann man komplexe Ereignisfolgen beschreiben. In Bild 4c ist eine Ereignisfolge und der zugehörige Ereignisstrom dargestellt: In einem Intervall der Dauer Null können bis zu drei Ereignisse auftreten, in einem Intervall der Dauer *j* treten maximal vier Ereignisse auf. Das ganze wiederholt sich periodisch mit einer Periode *p*. Der resultierende Ereignisstrom lautet somit $\theta_3=\{(p,0),(p,0),(p,0),(p,j)\}$ oder $\theta_3=\{3(p,0),(p,j)\}$. Die maximale Dichte der Ereignisse für gegebene Zeitintervalle Δt kann dann mit der Ereignisdichtefunktion berechnet werden:

$$ESF(\Delta t, \Theta) = \sum_{i \in \Theta} \left\lfloor \frac{\Delta t - a_i}{p_i} + 1 \right\rfloor \tag{3}$$

In Bild 4 ist diese Funktion für das Beispiel in Bild 4b graphisch dargestellt.

Mit Hilfe dieser Funktion kann leicht überprüft werden, ob ein mit diesem Ereignisstrom angestoßener Prozeß seine Zeitschranken einhält. dazu muß die Funktion $ESF(\Delta t, \Theta)$ mit der maximalen Ausführungszeit c_j des Prozesses *j* multipliziert und um die Frist d_j (Deadline) nach rechts verschoben werden:

$$DBF(\Delta t, \Theta) = \sum_{i \in \Theta} \left\lfloor \frac{\Delta t - a_i - d_j}{p_i} + 1 \right\rfloor \cdot c_j \tag{4}$$

Diese Rechenzeitanforderungsfunktion (*DBF*, demand bound funktion) genannte Abbildung beschreibt wieviel Rechenzeit im Zeitintervall Δt abgearbeitet sein muß. Damit ist ein System unabhängiger Prozesse echtzeitfähig, d.h. es hält alle an die Prozesse gesetzten Fristen ein, wenn

gilt:

$$\sum_{i \in \Theta} \left\lfloor \frac{\Delta t - a_i - d_j}{p_i} + 1 \right\rfloor \cdot c_j \le \Delta t \qquad (5)$$

Das bedeutet die Rechenzeitanforderungsfunktion muß immer unterhalb der Winkelhalbierenden liegen.

2.3.2 Hierarchische Ereignisströme

Mit Hilfe der Ereignisströme lassen sich komplexe Ereignisfolgen elegant beschreiben. Das Model hat jedoch einen Nachteil, Ereignisfolgen die Ereignisschübe (Bursts) beinhalten lassen sich nur mit viel Aufwand modellieren. In Gleichung (5) ist zu erkennen, dass der Berechnungsaufwand abhängig von der Anzahl der Ereignisstromtupel ist. In Bild 4 ist beispielhaft eine Ereignisfolge mit Schüben dargestellt. Zunächst tritt nach einer Zeit *x* jeweils ein Ereignis auf, dies erfolgt exakt sechs mal, danach tritt eine Pause auf und ein neuer Ereignisschub wird nach einer Zeit *y* nach Beginn des ersten Schubs ausgelöst. Um dieses Verhalten zu beschreiben benötigt man einen Ereignisstrom mit 6 Tupeln: Θ_4 = *{(p,0), (p,x), (p,2x),(p,3x), (p ,4x), (p ,5x)}*.

Da Ereignisschübe in Kommunikationssystemen und Rechnernetzen sehr häufig auftreten benötigt man bei der Modellierung dieser Systeme sehr häufig Ereignisschübe. Allerding treten Ereignisschübe nicht nur in Kommunikationssystemen auf, in jedem beliebigen eingebetteten Echtzeitsystem mit Datenabhängigkeiten zwischen den Prozessen können Ereignisschübe auftreten, und zwar dann, wenn von einem Prozeß Ereignisse in einer Schleife erzeugt werden. Möchte man nun die Anzahl der generierten Ereignisse in Abhängigkeit von der Anzahl der Schleifendurchläufe ausdrücken ist dies mit dem klassischen Ereignisströmen nicht möglich, da für jeden Schleifendurchlauf ein neues Tupel im Ereignisstrom erzeugt werden muß. Da dieses Problem bei verschachtelten Schleifen, die in realen Anwendungen häufig vorkommen, zu einer Explosion der Ereignistupel führt, wird ein Ereignismodell benötigt, indem die Anzahl der Schleifendurchläufe zur Signalerzeugung keine Veränderung der Struktur des Ereignismodells notwendig ist. Ein solches Modell sind die hierarchischen Ereignisströme. Dazu wird das Ereignistupel zu einem Quadtrupel erweitert:

$$\Theta' = \begin{pmatrix} p & n \\ a & \Theta \end{pmatrix} \qquad (6)$$

Dabei ist *p* weiterhin die Periode, *a* jetzt aber ein Zeitintervall, dass beschreibt, wann der Ereignisschub auftritt. In Θ' kann jetzt ein beliebiger hierarchischer Ereignis-strom eingebettet werden. Dabei begrenzt n die

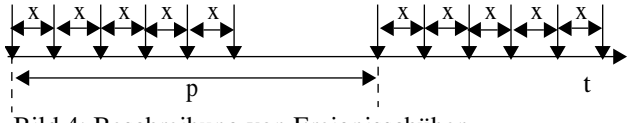

Bild 4: Beschreibung von Ereignisschüben

Anzahl der von diesem eingebetteten Ereignisstrom betrachteten Ereignisse. Mit *e:=(∞,0)* als Ereignisgrundelement ergibt sich folgender Zusammenhang

$$\begin{pmatrix} p & 1 \\ a & e \end{pmatrix} = \begin{pmatrix} p \\ a \end{pmatrix} \qquad (7)$$

zwischen den hierarchischen und den klassischen Ereignisströmen. Für die Analyse der Echtzeiteigenschaften eines Systems kann man die Ereignisdichtefunktion für die hierarchischen Ereignisströme modifizieren:

$$ESF(\Delta t, \Theta) = \begin{cases} 1 & \Theta' = e \\ \sum_{\varpi \in \Theta} \left(\left\lfloor \frac{\Delta t - a_\varpi}{p_\varpi} \right\rfloor \cdot n_\omega + R(\Delta t, \varpi) \right) & else \end{cases} \qquad (8)$$

$$R(\Delta t, \varpi) = min(n, ESF(((\Delta t - a_\varpi) mod\ p_\varpi), \Theta_\varpi'))$$

Damit diese Funktion gültig ist muß die Separationsbedingung für den hierarchischen Ereignisstrom erfüllt sein:

$$min(\Delta t | (EBF(\Delta t, \Theta) = n)) \le p_\omega$$

Hierarchische Ereignisströme, dieser Bedingung nicht genügen können leicht in solche Ereignisströme umgewandelt werden, die ihr genügen.

2.4. Ereignisabhängigkeitsgraphen

Ein Ereignisabhängigkeitsgraph (vgl. auch Bild 5a) ist ein Taskgraph an dem sowohl an die Knoten als auch an die Kanten ein Ereignisstrom annotiert wird. Dabei stellt der Ereignisstrom eines Knotens oder Prozesses die maximale Ereignisdichte, der von diesem Prozeß beim einmaligem Auslösen erzeugten Ereignisse dar. Der Ereignisstrom der Kanten modelliert die Kommunikationsereignisdichte zwischen den Prozessen. Sind die Ereignisströme an den Wurzeln des Ereignisabhängigkeitsgraphen durch die Umgebung des Systems und somit durch die Spezifikation gegeben, müssen die Ereignisströme, die von den Prozessen erzeugt werden, berechnet werden. Dies erfolgt mit Hilfe der Ereignisabhängigkeitsanalyse.

3. Transaktionsanalyse

3.1. Analyseablauf

Um eine Ereignisabhängigkeitsanalyse durchführen zu können (Bild 5b bis Bild 5d) muß das Programm eines Prozesses zunächst durch einen Compiler in den korrespondieren Flußgraphen umgewandelt werden (Bild 5b). Dies kann mit einem handelsüblichen Compiler-Frontend erfolgen. Allerdings muß der Compiler dahingehend erweitert werden, dass er Funktionen des Programms, die Ereignisse auslösen, erkennt und im Flußgraphen markiert.

Liegt ein solcher Flußgraph vor werden die minimalen Ausführungszeiten der einzelnen Basisblöcke im

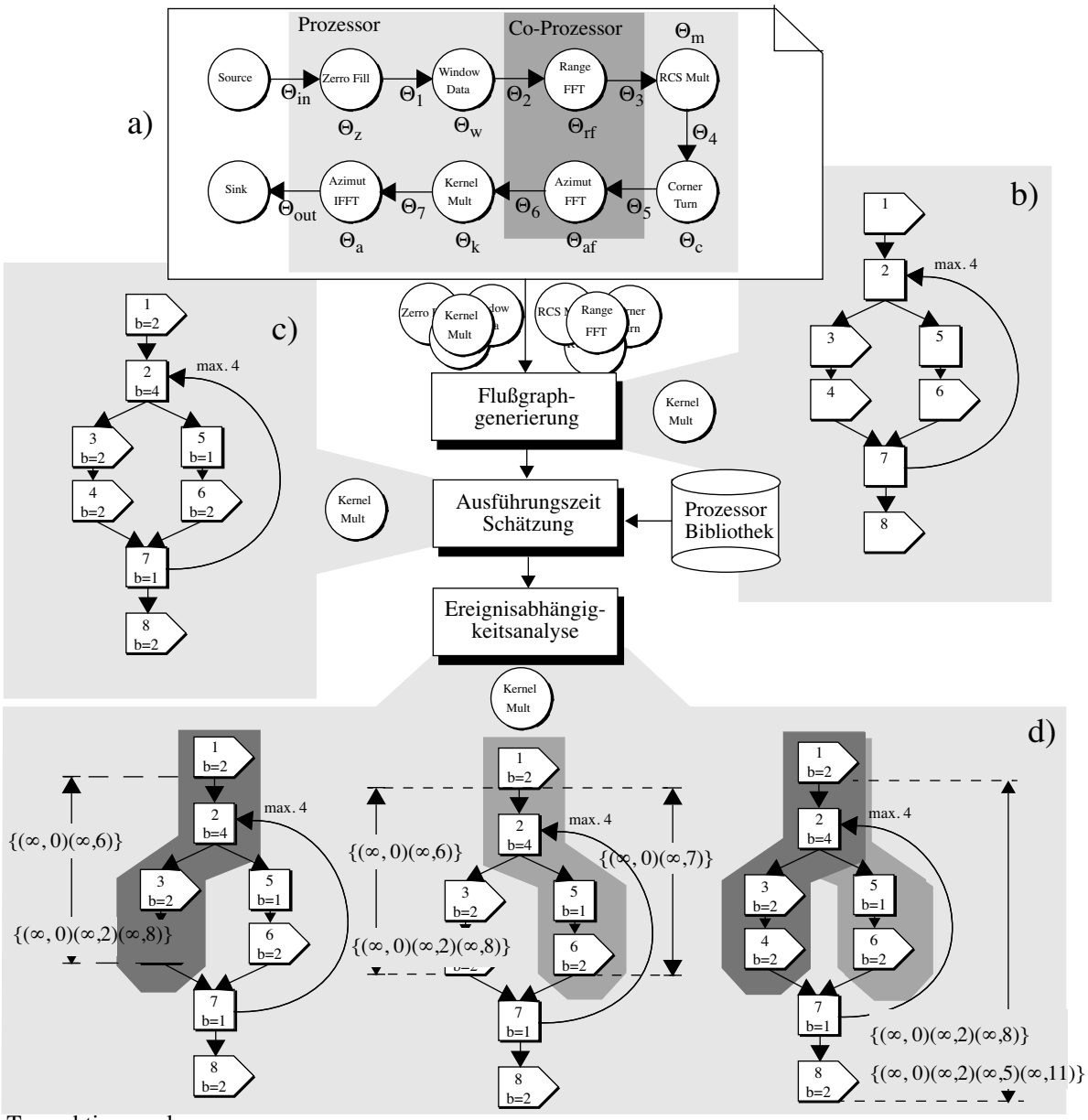

Bild 5: Transaktionsanalyse

Flußgraphen bestimmt (Bild 5c). Dies kann entweder mit einer Messung (Profiling) am realen System, einer Instruktionssatz-Simulation oder einem Schätzalgorithmus erfolgen. Im Anschluß wird dann die eigentliche Ereignisabhängigkeitsanalyse durchgeführt.

3.2. Ereignisabhängigkeitsanalyse

Ziel der Ereignisabhängigkeitsanalyse ist es die maximale Dichte von, von einem Prozeß generierten Ereignissen zu finden. Ausgehend von einem erweiterten Flußgraphen mit annotierten minimalen Ausführungszeiten der Basisblöcke werden die kürzesten Zeiten zwischen einer gegebenen Anzahl von Ereignissen durch eine Traversion des Flußgraphen ermittelt. Um den dichtesten Ereignisstrom mit einer Graphtraversion bestimmen zu können werden nur zwei Operationen auf dem Flußgraphen benötigt: Die Operation *concatenate* erlaubt es zwei Graphen linear aneinander zu hängen und so Schritt für Schritt die minimalen Zeitintervalle zu ermitteln. Mit der Operation *alternativ* werden Verzweigungen analysiert, in diesem Fall an der Stelle an der zwei getrennte Ausführungspfade wieder zusammenfließen. Daraus ergibt sich das folgende algorithmische Vorgehen:

- Traversiere den Flussgraphen Schritt für Schritt.
- Berechne in jedem Schritt den Ereignisstrom und zusätzliche Ereignissequenzen, die ggf. für die Berechnung des resultierenden Ereignisstromes benötigt werden.

In Bild 5d bis wird dieser Ablauf an einem Beispiel erläutert und Bild 6 zeigt den Traversionsalgorithmus. Um das Beispiel einfach zu halten wird die Schleife zunächst nicht betrachtet. Zuerst wird von der Wurzel aus das erste Ereignis gesucht. Für jeden Pfad zu einem weiteren Ereignis werden die Ausführungszeiten bis zu diesem Ereignis jeweils aufsummiert. Das wiederholt sich für jeden Pfad einer Verzweigung für die Ereignisanzahl *2,3,4,..., n*, bis die unterschiedlichen Pfade wieder zusammenlaufen. Im Beispiel wird im linken Pfad das erste Ereignis nach 6 Zeitschritten besucht, das dritte Ereignis nach 8 Zeitschritten. Das ergibt im ersten Schritt den folgenden Ereignisstrom $\theta_{kl}=\{(\infty,0),(\infty,6)\}$ und im zweiten Schritt den Strom $\theta_{kl}=\{(\infty,0),(\infty,2),(\infty,8)\}$, da Ereignis 2 und 3 enger zusammen liegen als Ereignis 1 und 2. Die Gleiche Analyse auf den rechten Pfad angewand ergibt $\theta_{kr}=\{(\infty,0),(\infty,7\}$. Mit der *merge* Operation kann man jetzt beide Pfade vereinigen. Da in einem Ereignisstrom die dichteste Ereignisfolge verwendet wird ist der resultierende Ereignisstrom $\theta_k=\{(\infty,0),(\infty,2),(\infty,8)\}$ für den Graphen bis *Knoten 7*. Nun wird einfach die Traversion fortgesetzt und *Knoten 8* berücksichtigt, da das Intervall von *Knoten 1* bis *Knoten 4* länger ist als das Intervall von *Knoten 3* bis *Knoten 8* ist die Zeit in der drei Ereignisse auftreten nur noch 5 Zeitschritte. Der resultierende Ereignisstrom ist also $\theta_k=\{(\infty,0),(\infty,2),(\infty,5)\}$. Betrachtet man nun die Schleife, die maximal 4 Mal durchlaufen wird erhält man den Ereignisstrom $\theta_k=\{(\infty,0),(\infty,2),(\infty,5),(\infty,12),(\infty,18),(\infty,23),(\infty,25),(\infty,30),(\infty,32),(\infty,38),(\infty,40)\}$.

Unter Verwendung der hierarchischen Ereignisströme läßt sich θ_k eleganter formulieren: In der Schleife treten maximal zwei Ereignisse auf, das führt zu dem Strom $\theta_{ks}=\{(\infty,0),(\infty,2)\}$. Die Schleife kann maximal 4 mal durchlaufen werden, also ergibt sich der folgende Ereignisstrom:

```
algorithm intervallsFromGraph (CFG G)
begin
   rollOutLoops(G);
   push (root(G));
   while stackNotEmpty do
      v := pop();
      if allPredesessorsVisited(v) then
         if numberOfPredessessors = 1 then
            Θ := analyze seq(predessosors(v),v);
         else if numberOfPredessessors > 1 then
            Θ:= analyze seq(alt(predessosors(v)),v);
         end if;
         markVisited(v);
         push(succsessor(v));
      end if;
   end while;
   return(Θ);
end;
```

Bild 6: Flussgraph-Analyse

$$\Theta_k = \left\{ \begin{pmatrix} \infty & 4 \\ 6 & \Theta_{ks} \end{pmatrix}, \begin{pmatrix} \infty & 1 \\ 5 & e \end{pmatrix}, \begin{pmatrix} \infty & 1 \\ 5 & e \end{pmatrix} \right\}$$

Der große Vorteil der hierarchischen Form liegt darin, dass sowohl Analyse als auch der Ereignisstrom einfacher sind. Möchte man statt vier Schleifendurchläufen die Schleife 2345 mal durchlaufen, kann man den hierarchischen Ereignisstrom direkt hinschreiben:

$$\Theta_k = \left\{ \begin{pmatrix} \infty & 2345 \\ 6 & \Theta_{ks} \end{pmatrix}, \begin{pmatrix} \infty & 1 \\ 5 & e \end{pmatrix}, \begin{pmatrix} \infty & 1 \\ 5 & e \end{pmatrix} \right\}$$

4.Entwurfsstudie

Bild 5 zeigt den Ereignisabhängigkeitsgraphen eines Synthetic Aperature Radar. Die Software eine Implementierung der Signalverarbeitung eines solchen zur Bodenerkundung eingesetzten Radargeräts ist als Benchmark des *RASSP* Projektes frei verfügbar und wird auch in anderen Arbeiten zu dem Thema als Benchmark eingesetzt [3], [5]. Man kann sich jetzt die Frage stellen, wie groß die Ereignisdichte auf den Verbindungsleitungen zwischen Standardprozessor und einem speziellen Coprozessor für die FFT-Prozesse des Radars ist. Zu diesem Zweck wurde eine Ereignisabhängigkeitsanalyse durchgeführt. Im klassischen Ereignisstrommodell werden über 2000 Tupel generiert. Der zugehörige hierarchische Ereignisstrom kann wie folgt geschrieben werden:

$$\Theta_1 = \left\{ \begin{matrix} \infty & 40960 \\ 0 & \Theta_2 \end{matrix} \right\}$$

$$\Theta_2 = \left\{ \begin{pmatrix} 6477 \\ 0 \end{pmatrix}, \begin{pmatrix} 6477 & 512 \\ 4.9 & \left\{ \begin{pmatrix} 9.58 \\ 0 \end{pmatrix} \right\} \end{pmatrix}, \begin{pmatrix} 6477 & 64 \\ 4898.6 & \left\{ \begin{pmatrix} 12.42 \\ 0 \end{pmatrix} \right\} \end{pmatrix}, \begin{pmatrix} 6477 & 63 \\ 5706 & \left\{ \begin{pmatrix} 12.42 \\ 0 \end{pmatrix} \right\} \end{pmatrix} \right\}$$

5.Zusammenfassung

Der Aufsatz beschreibt eine neue Methode das Transaktionsverhalten eingebetteter Echtzeitsysteme analytisch zu beschreiben und diese Beschreibung automatisch aus der Implementierung der Systemprozesse abzuleiten.

6.References

[1] Karsten Albers, Frank Bodmann, Frank Slomka. *Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems*. IEEE Proceedings of the 18th Euromicro Conference on Real-Time Systems, 2006.

[2] K. Richter and R. Ernst. *Event model interfaces for heterogeneous system analysis*. Proceedings of the Design, Automation and Test in Europe Conference (DATE'02), 2002.

[3] S. Goddard, X. Liu. *A Variable Rate Execution Model*. Proceedings of the 16th Euromicro Conference on Real-Time Systems, 2004

[4] K. Gresser. *An event model for deadline verification of hard real-time systems*. In 5th Euromicro Workshop on Real-Time Systems, Finland, 1993.

[5] K. Jeffay, S. Goddard. *A Theory of Rate-Based Execution*. Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.