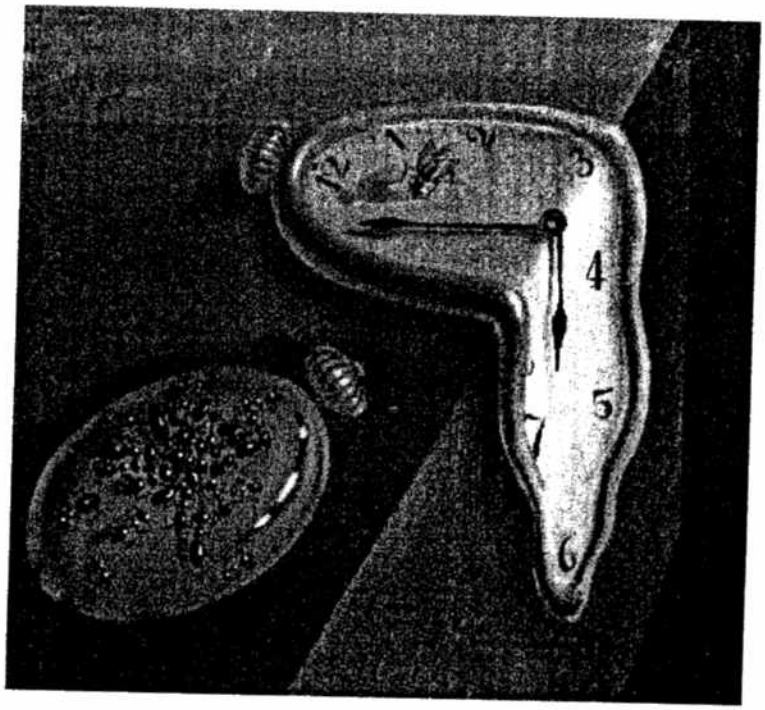
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1-18
April 1991

Monitoring in verteilten Echtzeitsystemen

U. Schmid



Ausschnitt aus: Salvador Dali, "Die Beständigkeit der Erinnerung"

Monitoring in verteilten Echtzeitsystemen

Ulrich Schmid*, TU Wien

Die vorliegenden Arbeit beschäftigt sich mit einem System für das Monitoring verteilter Echtzeitsysteme, das zur Zeit an der TU Wien entwickelt wird. Dieser *VTA* (*Versatile Timing Analyzer*) dient zur statistischen Auswertung des zeitlichen Auftretens beliebig vieler, frei definierbarer Ereignisse (*Events*) in dem zugrundeliegenden Echtzeitsystem. Eine Reihe flexibler Mechanismen zur Definition von Events und den darauf aufbauenden Meßgrößen (*Quantities*) erlaubt es, den VTA sowohl als Instrument des praktischen Requirement Engineerings als auch als universelles Testwerkzeug einzusetzen.

1. Einleitung

Ein Reihe sehr unangenehmer und bei weitem nicht gelöster Probleme im Zusammenhang mit dem Design von *ereignisgesteuerten Echtzeitsystemen* resultieren aus der mangelnden Beherrschbarkeit gewisser dynamischer Größen. Von zentraler Bedeutung ist etwa das Zeitverhalten derartiger Systeme, genauer gesagt, deren Fähigkeit, auf Zustandsänderungen in dem zu kontrollierenden *technischen Prozeß* innerhalb gewisser *Zeitschranken* zu reagieren. Übliche Multitasking–Systeme (zum Beispiel auf der Basis von os9 oder pSOS) leiden etwa sehr häufig darunter, daß zeitliche/funktionale Abhängigkeiten der Tasks untereinander zu unzulässig langen Bearbeitungszeiten führen können. Solche Situationen sind jedoch nicht immer leicht zu erkennen; die Anzahl der möglichen "Exekutionsreihenfolgen" eines Multitasking–Systems überschreitet ja bei weitem jene Größenordnung, die noch einen vollständigen Test erlauben würde.

Abgesehen von derartigen Problemen ist es auch schon sehr schwierig, das Zeitverhalten eines Echtzeitsystems im regulären Betrieb detaillierter zu erfassen. Fragen nach den zur Verfügung stehenden "Zeitreserven", also letztlich nach der Dimensionierung der notwendigen Rechnerleistung, werden im Normalfall äußerst pragmatisch beantwortet: Wenn das fertige System einige Zeit funktioniert, dann ist es auch richtig dimensioniert!

Eine ähnliche, sogar noch wesentlich tiefergehende Problematik ist im Zusammenhang mit den *Lasthypothesen* für ereignisgesteuerte Echtzeitsysteme zu orten. Dabei geht es um die Bereitstellung von realistischen theoretischen Modellen, die schon in einer frühen Phase der Systementwicklung Aussagen über die nötige Rechnerleistung ermöglichen. "Klassische" Performance–Maße wie die mittlere CPU–Auslastung sind in Hinblick auf die Einhaltung harter Zeitschranken ja leider relativ bedeutungslos.

Soferne nun die aus dem technischen Prozeß kommenden Signale nicht zyklisch oder sonstwie deterministisch auftreten, sind die heutzutage üblichen Angaben wie "im Mittel 4000 Ereignisse/Sekunde" (allein) ziemlich wertlos. Zusätzliche Informationen, etwa Wahrscheinlichkeitsverteilungen und Angaben über die Korrelation verschiedener Prozeßsignale, sind aber oftmals nicht verfügbar oder in der Praxis nicht verwertbar. Es erscheint uns daher zunächst erforderlich, relevante Kenngrößen jener Stimuli zu definieren und zu messen, mit denen technische Prozesse ein Echtzeitsystem konfrontieren. Aufbauend auf einem vernünftigen theoretischen Modell sollte es dann möglich sein, halbwegs realistische und trotzdem handhabbare Lasthypothesen zu formulieren.

Der zur Zeit am Institut für Automation (183/1) der TU Wien in Entwicklung befindliche *VTA* (*Versatile Timing Analyzer*) ist nun ein Werkzeug, mit dessen Hilfe dem umrissenen Problemkreis praktisch zu Leibe gerückt werden kann. Grob klassifiziert handelt sich hierbei um ein System zur (statistischen) Auswertung des zeitlichen Auftretens frei definierbarer *Events* in einem verteilten Echtzeitsystem.

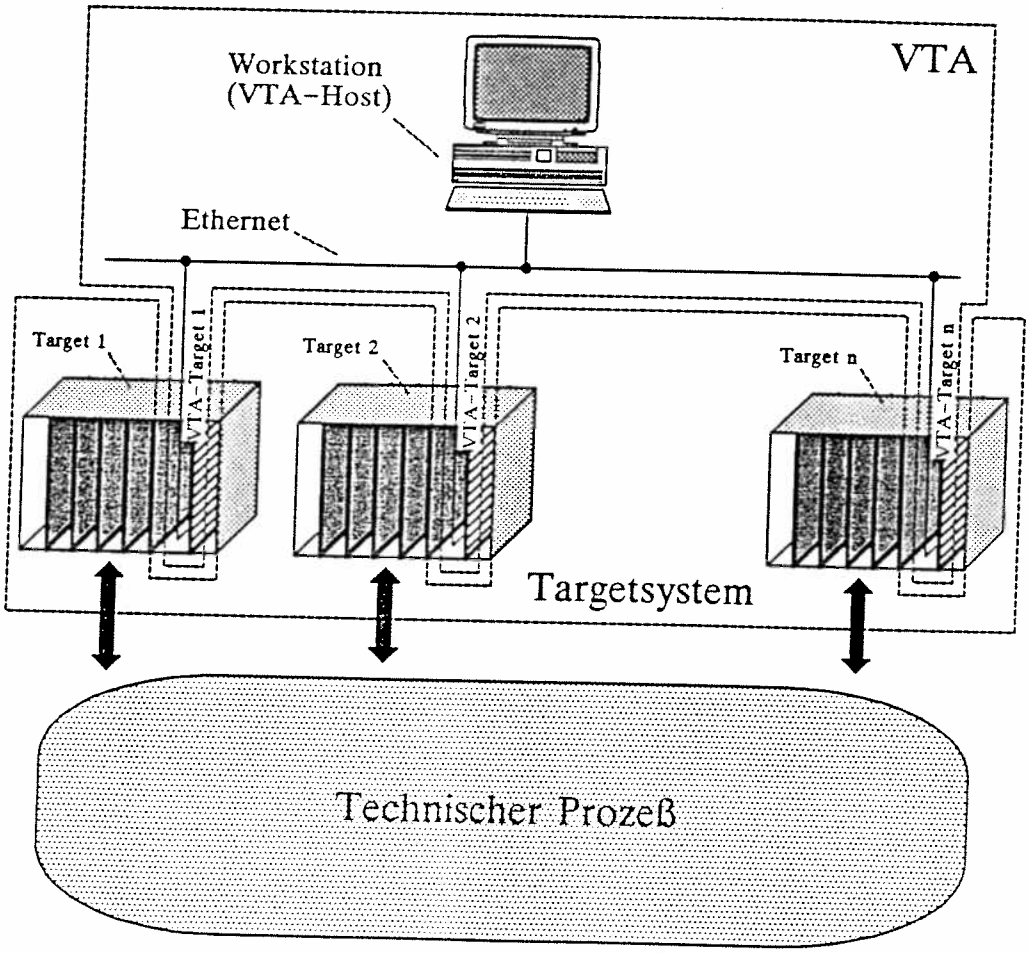
* TU Wien, Institut für Automation (183/1), Treitlstraße 3, A–1040 Wien (Tel. 0222/58801–8189)

Die folgenden Abschnitte 2 (*Konfiguration*) und 3 (*Funktionalität*) sind der Vorstellung des prinzipiellen Aufbaus und der Funktionalität des VTAs gewidmet, Aspekte der Implementierung werden im Abschnitt 4 (*Realisierung*) erläutert. Einige abschließende Bemerkungen finden sich schließlich im Abschnitt 5 (*Ausblick*).

2. Konfiguration

Der projektierte Prototyp des VTAs ist für das Monitoring in einem verteilten VME-basierenden Echtzeitsystem (im folgenden *Targetsystem* genannt) gedacht. Ein solches Targetsystem besteht aus einem oder mehreren, untereinander beliebig gekoppelten VME-Racks mit jeweils einer oder mehreren CPUs. Das offene Design des VTAs sieht die prinzipielle Unterstützung verschiedener (Target-)Programmiersprachen (C, PEARL, Ada, ...), verschiedener Betriebssysteme (pSOS, VRTX, os9, ...) und verschiedener CPU-Boards bzw. Prozessoren vor.

Der VTA selbst besteht aus einer oder mehreren Monitoring-CPUs (*VTA-Targets*), die in die einzelnen Racks des Targetsystems gesteckt werden, und einer Workstation (*VTA-Host*). Diese Komponenten werden durch ein Ethernet-basierendes LAN miteinander verbunden. Das folgende Bild zeigt den prinzipiellen Aufbau:



Der VTA-Host stellt das Benutzer-Interface des VTAs bereit; zur Laufzeit des Targetsystems können Monitoring-Funktionen aktiviert und deaktiviert und erfaßte Daten (*Quantities*) ausgewertet bzw. dargestellt werden. Den VTA-Targets obliegt hingegen das Setzen/Modifizieren/Löschen von Instrumentierungspunkten in der Targetsoftware und natürlich die Erfassung und Vorverarbeitung der über diese Instrumentierung "signalisierten" Events.

3. Funktionalität

Von zentraler Bedeutung ist der Begriff eines *Events*, das eine bestimmte Situation im Targetsystem spezifiziert. Das Eintreten der durch ein Event charakterisierten Situation wird als *Occurrence* bezeichnet. Unter dem *Aufsetzen* eines Events ist die konkrete Formulierung einer solchen Situation und die Festlegung der Aktionen, die bei der Occurrence des entsprechenden Events ablaufen sollen, zu verstehen. Jedes Event wird durch einen *Event Name* identifiziert, der im allgemeinen beim Aufsetzen des jeweiligen Events zu vergeben ist. Die elementaren Vertreter von Events sind die *Simple Events*:

(1) *Statement Simple Events*
Ein derartiges Simple Event tritt ein, wenn ein bestimmter Befehl in der Targetsoftware exekutiert wird. Der VTA erlaubt das Aufsetzen von Statement Events auf Hochsprachenebene, rechnet also einerseits mit dem Vorhandensein der entsprechenden Sourcefiles am VTA–Host und andererseits mit einer "modifizierbaren" (im RAM befindlichen) Software im Targetsystem. Es ist aber auch möglich, gewisse (vor allem extrem zeitkritische) Events schon bei der Entwicklung der Targetsoftware einzubinden.

(2) *Special Simple Events*
Bei derartigen Simple Events handelt es sich um spezielle Ereignisse wie *System Call Entry*, *System Call Exit* oder das *Dispatching*. Mit einigem zusätzlichen Hardware–Aufwand können (später) auch auch Simple Events betreffend gewisse (Bus–)Signale, vor allem Interrupt–Leitungen, bereitgestellt werden.

(3) *Time Simple Events*
Hier können beliebige Zeitpunkte (zyklisch oder one–shot) aufgesetzt werden, bei deren Erreichen das korrespondierende Simple Event eintritt.

(4) *Quantity Simple Events*
Ein derartiges Simple Event tritt ein, wenn der Wert einer Quantity eine spezifizierte Bedingung erfüllt.

In allen Fällen können dabei auch Nebenbedingungen gefordert werden. Ein *Conditional Simple Event* tritt nur dann ein, wenn das Simple Event eintritt und die spezifizierten Nebenbedingungen zu diesem Zeitpunkt erfüllt sind. Im Detail existieren hier unter anderem folgende Möglichkeiten:

- *Exekution im Kontext eines oder mehrerer bestimmter Tasks/Prozessoren*
- *Vorliegen bestimmter Variablenwerte*
- *Vorliegen bestimmter Werte in den Prozessor–Registern*
- *Nebenbedingungen mit VTA–Variablen* (letztere werden im VTA organisiert)

Ein *Event* stellt nun eine Verknüpfung von (Simple) Events dar. Solche Verknüpfungen werden werden mit Hilfe einer geeigneten *Event Definition Language* (*EDL*) formuliert, siehe dazu auch **Bat83**. Die Syntax unserer EDL folgt der herkömmlichen Notation der geklammerten Ausdrücke mit +, – und ·, wobei natürlich die Semantik der Operatoren unterschiedlich ist. Ein Beispiel, nämlich das Event *"Telefonat mit einem Familienmitglied"*, soll dies illustrieren:

E := ABHEBEN·WÄHLEN·(EHEFRAU+KIND–BESETZT)·SPRECHEN·AUFLEGEN

Das zitierte Ereignis tritt nur dann ein, wenn zuerst die Events ABHEBEN und dann WÄHLEN eintreten und danach entweder die EHEFRAU oder das KIND den Hörer abnehmen. Sollte hingegen statt dessen das Event BESETZT eintreten, wird wieder der Status des Beginns (also vor dem ABHEBEN) erreicht.

Die in einem derartigen Ausdruck verwendeten Events dürfen nun Simple Events oder aber auch andere Events sein; das Event SPRECHEN könnte etwa durch

SPRECHEN := SATZ_SAGEN·(VERSTANDEN–NICHT_VERSTANDEN)·SERVUS

aufgesetzt werden. Das Event NICHT_VERSTANDEN würde hier den Zustand vor dem SATZ_SAGEN wiederherstellen.

Beim Aufsetzen eines Events kann spezifiziert werden, welche Aktionen bei der Occurrence desselben ausgeführt werden sollen. Es gibt dabei zwei Klassen von Aktivitäten:

(1) *Aktivitäten im Targetsystem auslösen*
Hierbei geht es hauptsächlich um die Modifikation bestimmter Variablen im Targetsystem.
(2) *Aktivitäten im VTA(–Target) auslösen*
Neben der Modifikation von VTA–Variablen können hier unter anderem Timer Simple Events aufgesetzt und Quantities verändert werden.

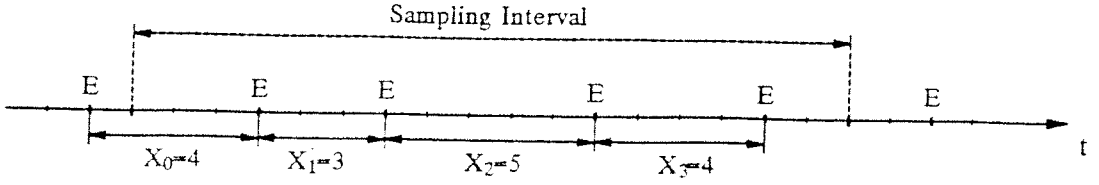
Wie bei den Simple Events können auch für ein Event als Ganzes Nebenbedingungen mit VTA–Variablen gefordert werden. Auf diese Weise ist zum Beispiel ein Suspend/Resume von Events über VTA–Variable zu realisieren.

Bei den erwähnten Möglichkeiten handelt es sich um das von der Funktionalität her gesehen mächtigste, aber klarerweise etwas umständlich zu handhabende *Low Level VTA–Programming*. Darauf aufbauend werden einige höhere Mechanismen bereitgestellt, die eine Reihe von häufig benötigten (Standard–)Meßverfahren (also Standard–Quantities) realisieren. Verzwickte Meßprobleme können aber selbstverständlich durch spezielle "(User–)Programme" auf der Ebene des Low Level VTA–Programmings gelöst werden.

Der VTA soll im Endeffekt bestimmte Parameter wie Maxima, Minima, Mittelwert, Varianz, usw. beliebig vieler, frei definierbarer *Quantities* (*Meßgrößen*) erfassen. Jede solche Quantity gehört einer bestimmten *Quantity Class* an. Betrachtet man das Targetsystem während eines gewissen *Sampling Intervals*, so werden in der Regel die eine Quantity bestimmenden Events mehrmals auftreten, sodaß am Ende eines Sampling Intervals eine ganze Folge von Meßwerten X_0, X_1, ..., X_n für die entsprechende Größe vorliegen wird.
Unter anderem sind nun folgende (Standard–)Quantity Classes, quasi ein "Basiskatalog" von Meßmöglichkeiten, denkbar:

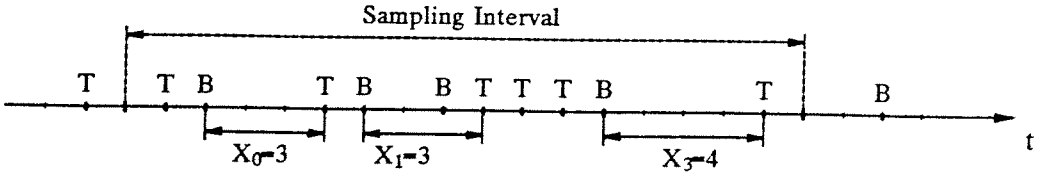
(1) *Durations*
Dabei handelt es sich im Prinzip um die Erfassung von mehr oder weniger komplexen Zeitintervallen.

(A) *Interarrival Times*
Eine Interarrival Time wird durch ein einzelnes Event E bestimmt:



(B) *Simple Durations*
Eine Simple Duration durch ein Begin–Event B und ein korrespondierendes Termination–Event T definiert. Hierbei gibt es

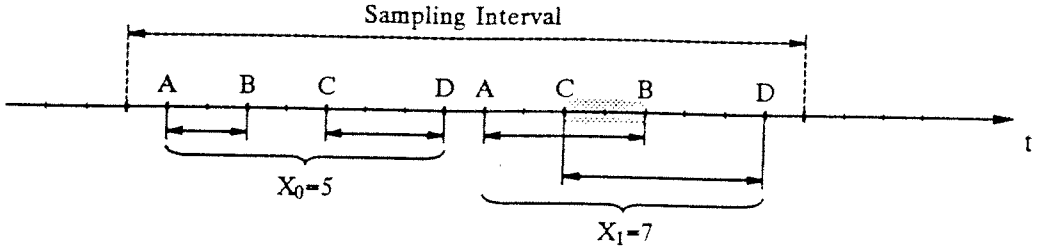
- *Single Asymmetric Simple Durations*
  Dies ist die (immer nicht–negative) Differenz zwischen der Occurrence von T und B, wobei B vor T kommen muß:



- *Single Symmetric Simple Durations*
  Dabei handelt es sich um die Differenz zwischen der Occurrence von T und B, wobei auch T vor B zugelassen wird. Eine negative Duration ergibt sich genau dann, wenn das Termination–Event vor dem Begin–Event kommt.
- *Multiple Asymmetric Simple Durations*
  Dies ist wieder die (immer nicht–negative) Differenz zwischen der Occurrence von T und B, wobei B vor T kommen muß. Wichtig ist, daß in diesem Fall zwei oder mehrere Durations gleichzeitig "aktiv" sein können. Kommen also zwei Begin–Events hintereinander, so werden gleichzeitig zwei Meßwerte (X_i und X_{i+1}) erfaßt.
- *Multiple Symmetric Simple Durations*
  Dabei handelt es sich wieder um die Differenz zwischen der Occurrence von T und B, wobei auch T vor B zugelassen wird. Hier können ebenfalls, wie zuvor, mehrere Durations gleichzeitig "aktiv" sein.

(C) *Single Complex Durations*
Derartige Quantities ergeben sich aus der Summe mehrerer (Single Asymmetric oder Single Symmetric) Simple Durations. Zu beachten ist, daß zu jedem Zeitpunkt höchstens eine solche Meßgröße aktiv sein kann. Es gibt jedoch zwei Varianten für den Fall, wo sich zwei beteiligte Simple Durations überlappen:

- *Overlapped Single Complex Durations*

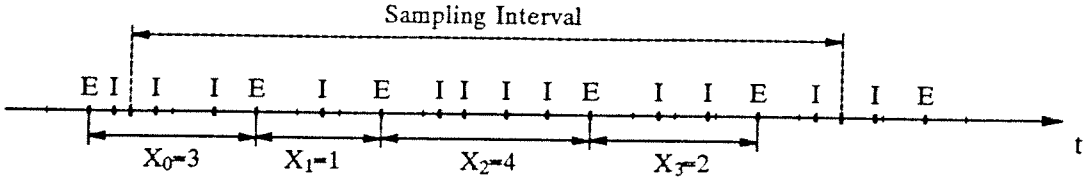


Überlappende Bereiche (schraffiert) werden nur einmal "gezählt".

- *Nonoverlapped Single Complex Durations*
  Überlappende Bereiche werden hier mehrfach "gezählt".

(2) *Duration Eventcounts*
Dabei handelt es sich im Prinzip um die Erfassung der Anzahl der Occurrences eines Events I innerhalb mehr oder weniger komplexer Zeitintervalle.

(A) *Interarrival Time Eventcounts*
Im Gegensatz zu der bereits vorgestellten Quantity Class Interarrival Times handelt es sich hier um die Erfassung der Anzahl von Events I zwischen den Occurrences eines Events E:



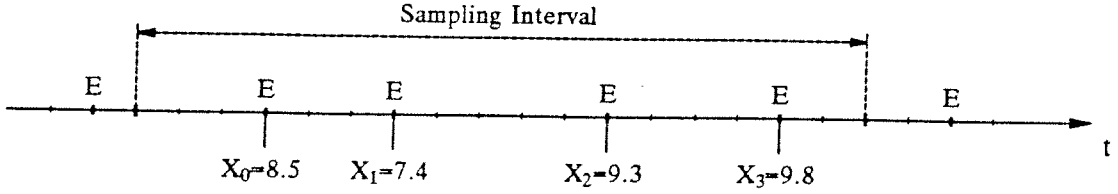
(B) *Single Simple Duration Eventcounts*
Eine Simple Duration wird bekanntlich durch ein Begin-Event B und ein korrespondierendes Termination-Event T definiert. Bei den darauf aufbauenden Eventcounts gibt es zwei Möglichkeiten:

- *Single Asymmetric Simple Duration Eventcounts*
  Dies ist die Anzahl der Occurrences von Event I zwischen Occurrences von T und B, wobei B vor T kommen muß.
- *Single Symmetric Simple Duration Eventcounts*
  Dabei handelt es sich um die Anzahl der Occurrences von I zwischen Occurrences von T und B, wobei auch T vor B zugelassen wird.

Zu beachten ist, daß zu jedem Zeitpunkt nur eine Meßgröße aktiv sein kann.

(C) *Single Complex Duration Eventcounts*
Derartige Quantities ergeben sich aus der Summe mehrerer (Single Asymmetric oder Single Symmetric) Simple Duration Eventcounts. Auch hier kann zu jedem Zeitpunkt höchstens eine solche Meßgröße aktiv sein.

(3) *Values*
Hierbei handelt es sich um Werte spezifizierter Variablen beim Eintreten eines Events.



4. Realisierung

Bei der Realisierung des VTAs sind drei voneinander relativ unabhängige Teilaspekte zu unterscheiden:

(1) *Bereitstellung eines leistungsfähigen verteilten Systems von VTA-Targets*
Die Aufgabenstellungen, mit der sich der Verbund der VTA-Targets konfrontiert sieht, erfordern zunächst einmal relativ leistungsfähige (single-board) Hard-

ware-Komponenten. Ein VTA-Target muß ja den "Strom" der aus dem jeweiligen Targetsystem stammenden Events entgegennehmen und entsprechend den Ausführungen im folgenden Punkt (2) verarbeiten. Die *Distributed Event Recognition* erfordert darüberhinaus auch einen verteilten(!) Timestamp-Mechanismus sehr feiner Granularität, was im Endeffekt auf das bekannte Problem der *Clock-Synchronisation* in verteilten Systemen führt (siehe zum Beispiel **Lam78, Kop87, Arv89**).

Die Realisierung der VTA-Targets erfolgt auf der Basis von Force CPU-30 Boards. Diese sollen jedoch anstelle des standardmäßigen Ethernet-Piggybacks ein spezielles Netzwerk-Interface tragen, das hardwaremäßig die Clock-Synchronisation ganz wesentlich erleichtert. Die Kommunikation der Instrumentierungsroutinen in der Targetsoftware mit dem jeweiligen VTA-Target wird über das Force Message Broadcast abgewickelt.

Als Betriebssystem werden wir pSOS+ von Software Components Group einsetzen. Systemnahe Funktionen wie die Clock-Synchronisation können hier im ohnedies notwendigen *Multiprocessor Interface* verborgen werden, sodaß sich im Endeffekt ohne viel Aufwand ein leistungsfähiges, clock-synchronisiertes verteiltes System von VTA-Targets bereitstellen läßt. Da auf der ohnedies für den VTA-Host benötigten Workstation auch gleich die Entwicklungsumgebung für das pSOS+ installiert werden kann, ergibt sich auch eine optimale Konfiguration für die Entwicklung der Software der VTA-Targets.

(2) *Realisierung der Software für die VTA-Targets*
Die Hauptaufgabe der VTA-Targets ist die *Distributed Event Recognition* auf Basis einer geeigneten Event Definition Language. Diese erlaubt die Spezifikation der interessierenden Events und impliziert dadurch auch eine Art Datenabstraktion, also letztlich eine Datenreduktion. Grundvoraussetzung dafür ist allerdings ein verteiler Algorithmus, der ein Matching der von den VTA-Targets erfaßten Simple Events mit den in der EDL formulierten komplexeren Events durchführt; ein bei weitem nichtriviales Problem (siehe **Bat83, Bat88, Spe88**).

Darüberhinaus müssen natürlich auch jene Möglichkeiten realisiert werden, die das Aufsetzen/Ändern/Löschen von Events erlauben. Von besonderer Bedeutung ist hier das Eintragen der für Statement Simple Events notwendigen Instrumentierungen in der Targetsoftware (siehe dazu auch **Hab90**).

(3) *Realisierung der Software für den VTA-Host*
Hierbei handelt es sich um eine "klassische" UNIX Workstation-Applikation, nämlich um die Bereitstellung der Benutzerschnittstelle in einer geeigneten Multiwindow-Technik. Das eigentliche Problem liegt jedoch nicht so sehr darin, wie etwa die Daten-Darstellung nun tatsächlich realisiert werden kann. Die zentrale Frage ist vielmehr die, welche Kenngrößen überhaupt von Interesse sind und wie aussagekräftige Darstellungen aussehen sollten. Es gibt zwar im Kontext der "gewöhnlichen" verteilten Systeme einige Ansätze dazu (siehe **Mcd89** für einen Überblick), für Echtzeitsysteme sind diese Ideen jedoch kaum verwendbar. Es scheint sich hier tatsächlich weitgehend um Neuland zu handeln.

5. Ausblick

Obwohl der VTA für uns primär ein Werkzeug darstellt, mit dessen Hilfe wir realistische "Eingangsdaten" für unsere theoretischen Untersuchungen zum Problemkreis *Lasthypothesen/Performance Requirements* für ereignisgesteuerte Systeme gewinnen wollen, war uns auch die unmittelbare Praxisrelevanz eines derartigen Meßsystems für den *Test von Echtzeitsystemen* von Anfang an bewußt. Das zugrundeliegende Konzept stellt

dabei einen vernünftigen Kompromiß zwischen vertretbarem Aufwand einerseits und zulässiger Beeinflussung der ablaufenden Targetsoftware andererseits dar.

Der VTA erlaubt es zum Beispiel, die Dauer einer zeitkritischen Befehlssequenz der Targetsoftware kontinuierlich zu überwachen und auf diese Weise Performance–Engpässe (oder Überdimensionierungen) aufzudecken. Die Besonderheit liegt dabei in der Möglichkeit, komplexe Bedingungen spezifizieren und auf diese Weise den "Scope" einer solchen Untersuchung zum Beispiel auf einen oder mehrere bestimmte Tasks einschränken zu können. Spezielle komplexe Quantities gestatten es sogar, Entscheidungen betreffend die Verteilung von Tasks auf verschiedene Prozessoren zu verifizieren. Selbst die Dimensionierung von Speicherplatz für dynamische Datenstrukturen (Buffer, Heaps, Stacks, ...) kann überprüft werden.

Im Zusammenhang mit dem Problemkreis *Lasthypothesen* eröffnet der VTA die Möglichkeit, hierfür notwendigen Kenngrößen praktisch zu bestimmen bzw. zu verifizieren. Es ist dazu lediglich notwendig, Events in den für die Bedienung der Prozeßperipherie zuständigen Programmteilen aufzusetzen und geeignete Quantities zu erfassen. Die Voraussetzung ist natürlich ein an dem jeweiligen technischen Prozeß angekoppeltes Targetsystem. Darüberhinaus können aber auch die "Auswirkungen" derartiger Stimuli, also letztlich die in der Targetsoftware ausgelösten weiteren (also weitergehenden) Aktivitäten, verfolgt werden. Auf eine derartige praktische "Beobachtung" realer Systeme kann in Hinblick auf eine vernünftige Modellbildung wohl kaum verzichtet werden. Allerdings erfordern derartige Anwendungen des VTAs mit Sicherheit erweiterte Meß– und Auswertungsmöglichkeiten; entsprechende Erfordernisse werden sich sicherlich erst im Laufe der Zeit ergeben.

Gerade in diesem Zusammenhang sollten wir abschließend anmerken, daß sich das Projekt VTA sicherlich noch einige Zeit in einer Design–Phase befinden wird, in der die tatsächliche Realisierung sekundär ist. In Anbetracht der spärlichen Ergebnisse unserer bisherigen Literaturrecherche sind wir gezwungen, einige grundsätzliche Probleme selbst in Angriff zu nehmen. So gibt es, wie schon erwähnt, zum Teil sehr brauchbare Literatur aus dem Kontext des Monitorings "gewöhnlicher" verteilter Systeme, die etwa bei der Clock–Synchronisation oder der Distributed Event Recognition hilfreich sind, aber keine für Echtzeitsysteme verwendbaren Ansätze im Bereich der Datendarstellung.

Literatur

Arv89 Arvind, K., *"A New Probabilistic Algorithm for Clock Synchronization"*, Proc. Real–Time Systems Symposium IEEE (1989), 330–339

Bat83 Bates, P., Wileden, J.C., *"High–Level Debugging of Distributed Systems: The Behavioral Abstraction Approach"*, Journal of Systems and Software 3, (1983), 255–264

Bat88 Bates, P., *"Debugging Heterogeneous Distributed Systems Using Event–Based Models of Behavior"*, Proc. ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging (1988), 11–22

Hab90 Haban, D., Wybranietz, D., *"A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems"*, IEEE Trans. Soft. Eng., Vol. 16, No. 2 (Feb. 1990), 197–211

Lam78 Lamport, L., *"Time, Clocks and the Ordering of Events in a Distributed System"*, Comm. ACM, Vol. 21, No. 7, (July 1978), 558–565

Kop87 Kopetz, H., Ochsenreiter, W., *"Clock Synchronization in Distributed Real Time Systems"*, IEEE Trans. Comput., Vol 36, No. 8, (August 1987), 933–940

Mcd89 McDowell, C.E., Helmbold, D.P., *"Debugging Concurrent Programs"*, ACM Comput. Surv., Vol. 21, No. 4, (December 1989), 593–622

Spe88 Spezialetti, M., Kearns, J.P., *"A General Approach to Recognizing Event Occurrences in Distributed Computations"*, Proc. 8th Int. Conf. on Distrib. Comp. Syst., (1988), 300–307