EINDHOVEN UNIVERSITY OF TECHNOLOGY

Distributed real-time systems : a survey of applications and a general design model

Citation for published version (APA):
Luit, E. J., Hammer, D. K., Aarts, E. H. L., & Gorp, van, P. C. N. (1997). *Distributed real-time systems : a survey of applications and a general design model*. (Computing science reports; Vol. 9710). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/1997

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 16. Nov. 2023

Eindhoven University of Technology
Department of Mathematics and Computing Science

Distributed real-time systems:
a survey of applications and a general design model

by

P.C.N. van Gorp, E.J. Luit, D.K. Hammer, E.H.L. Aarts

97/10

ISSN 0926-4515

All rights reserved
editors: prof.dr. R.C. Backhouse
prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Reports 97/10
Eindhoven, June 1997

Distributed real-time systems: a survey of applications and a general design model

P.C.N. van Gorp[1], E.J. Luit[1], D.K. Hammer[1], E.H.L. Aarts[12]

[1] Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands

[2] Philips Research Laboratories, Eindhoven, The Netherlands

Abstract

Computer-controlled dependable systems play a crucial role in modern society. These systems can be encountered in many application domains, from multimedia systems to food processing machines.

The design of dependable real-time systems is very complex and calls for computer aided support. Scheduling is one of the important stages in the design trajectory. The input of the scheduler should describe all relevant features of the application.

The aims of this paper are twofold. Firstly, a survey is made of relevant timing characteristics in applications from different domains. Secondly, a model is presented which reflects all important concepts of dependable real-time systems (such as replication and communication). The model incorporates all the timing characteristics which are discussed in the survey. It is shown how different temporal behaviors can be expressed in the model.

The model is used in the design of an off-line hard real-time scheduler for the DEDOS system [8], which is being developed at the Eindhoven University of Technology.

1 Introduction

The DEpendable Distributed Operating System (DEDOS) project [8] is concerned with dependable distributed real-time systems. The development of techniques and algorithms for the design of these systems is the main purpose of the project. Nowadays, the use of dependable distributed real-time systems is growing fast, for example in the field of embedded systems. An important aspect of dependable systems is timeliness; often tasks have to be completed before a given deadline. Another important aspect of dependability is reliability; safety critical applications require that the system continues operation despite hardware failures. A nuclear plant and an air traffic control system are good examples where these issues play an important role. Other aspects of dependability are safety, availability, security and robustness. In general, it is very difficult to achieve all these conflicting goals simultaneously. At the moment, DEDOS supports timeliness and reliability.

With respect to timeliness one can distinguish two areas of interest. In soft real-time scheduling, tasks have to be completed before their deadlines as much as possible, but it is not catastrophic for a task to finish after its deadline. In hard real-time scheduling this is not allowed: each task has to finish before its deadline.

DEDOS supports applications with both hard real-time (HRT) and soft real-time (SRT) tasks. The hard real-time part forms the backbone. Hard real-time tasks are scheduled first, while soft-real time tasks are scheduled on idle resources in the remaining time. To guarantee timeliness, hard real-time tasks are scheduled statically. So, it is necessary that all timing aspects of these tasks are known beforehand. In some cases, worst case estimates have to be made to this end, which goes at the cost of resource utilization. Static scheduling is specifically suitable for applications which are characterized by periodicity.

Each application domain has its own timing characteristics. To be able to make an abstract formalization of real-time systems, it is important to identify the relevant timing behavior of these systems. This paper contains a survey of the application domains of dependable distributed real-time systems. Furthermore, a general model is presented, in which these systems can be formalized.

In Section 2, some characteristic HRT constraints are discussed and some examples from different application domains are given. In Section 3, it is shown how timing constraints can be expressed by so-called timed precedence constraints (TPCs). A model which incorporates the aspects of timeliness, reliability and hard real-time scheduling is presented in Section 4. Section 5 contains conclusions and in Section 6 some issues with respect to future research are discussed.

2 Survey of timing constrains

Hard real-time demands can be encountered in many types of applications. In this section a brief overview is given of different application domains that establish (periodic) scheduling problems. Special attention is paid to typical timing constraints for each domain.

2.1 Multimedia

Multimedia systems usually employ four components: speech, sound, graphics and video. Each of these components has its own timing characteristics. In video and audio applications, algorithms have to be performed on incoming data. Execution graphs represent the algorithms [6][10]. The operations in the algorithms are the nodes in the execution graphs. Operations are connected by data dependencies. These data dependencies define precedences, because correct functionality requires that data is produced by an operation before it is consumed by a different operation.

The algorithms have a periodic behavior, because the input data arrives periodically. In video applications and digital signal processing (DSP) in particular, the frequency of incoming data is much higher than in audio applications. Moreover, the latency between the input of data and the output is data is larger, because of the greater complexity of the video algorithms. Therefore, periodicity constraints are much stricter in video and DSP applications. DSP algorithms are used for applications such as picture in picture, Moire transformation and contrast adjustment.

Periodicity can be encountered in many places in DSP applications. For example, television images are constructed by periodically projecting pixels, i.e. an operation is performed for each pixel in a row and for each row in a frame at a certain (predefined) rate or period [15] (for example, 64 μs per line and 74.1 ns between consecutive pixels in one line). This period implicitly defines a deadline for each execution of the operation.

Another example of periodicity is the decoding of scrambled information used in the D2MAC standard. Every 10 seconds a new decoding scheme is provided. This introduces a periodic timing constraint for this task [17].

Other timing constraints arise from the interactive part of multimedia systems. Although the interactions between the system and the user typically have soft real-time characteristics, also stricter timing constraints have to be met as well. For example, if the user switches to a different channel on his or her television set, audio is muted first for 100 ms and a steady picture of the chosen channel is not displayed for at least 300 ms. When a video recorder is turned on, the screen must be blanked for at least 150 *ms*. Also, maximum response times to remote control actions are usually prescribed for the design.

2.2 Flexible manufacturing

A flexible manufacturing system (FMS) is characterized by the production of small batches of products. Products in different batches are similar, but not identical. Changes between different batches require short configuration switch times (set-up times). This is in contrast to the dedicated assembly lines used for mass production.

A FMS consists of a group of machines and a transportation mechanism to move the objects between these machines. Each machine is capable of performing a number of different operations. A robot is an example of such a flexible machine. Robotics is discussed in Section 2.2.1.

The objects under consideration do not always have the same outlook or appearance, and therefore require different (types of) operations. To this end, a FMS is usually provided with a sensing mechanisms (cameras, pressure valves, thermometers etc.). Computer vision and automated visual inspection are discussed in Section 2.2.2, while the control of sensors is the subject of Section 2.2.3.

2.2.1 Robotics

One way to establish the control of a robot is to calculate the movements of the joints of the robot. It is possible to partition this computation into a number of tasks [2]. Any statement that would normally cause such a computation to block is known as a (re)scheduling point and forms a task boundary. Depending on the partitioning, precedence relations exist on the execution of different tasks. These tasks are the unit of scheduling; they should be assigned to processors (in the controlling system) and the sequence of executions of tasks on the same processor should be determined. Furthermore, the robot-motion controlling mechanism shows a periodic behavior, and deadlines can be given for groups of tasks.

The problem of computing robot motions can be formulated as a multiprocessor scheduling problem, to minimize the overall execution time, where the tasks with different execution times have precedence relations and are allocated to a number of processors [9].

2.2.2 Computer vision

Computer vision deals with the recognition of structures or objects in a scene, within a restricted amount of time. Examples include part and position recognition, weather analysis, analysis of currents of fluids, and molecular dynamics. Typical input for these processes is a continuous row of frames (for example, at a rate of 30 frames a second) with 1024×1024 pixels in each frame. In general, many operations have to be performed on each pixel. Because output has to be produced within a reasonable amount of time, this could lead to $10^{11} - 10^{13}$ operations per second. So, a high throughput has to be achieved using parallel processing.

One application of computer vision is automated visual inspection (AVI) [12]. Objects are moved on a conveyor belt for inspection (figure 1). Depending on the result of the inspection, some operation may be performed on the objects (for example the removal of cherry-stones from cherries or the filleting of fish in food processing industries).

Suppose, for the moment, that incorrect objects need to be taken off the belt. For this, a camera captures images from the objects on the belt. These images are input to the decision procedure, which determines if the scanned object should be removed. If so, the object will be removed from the belt by some mechanism.

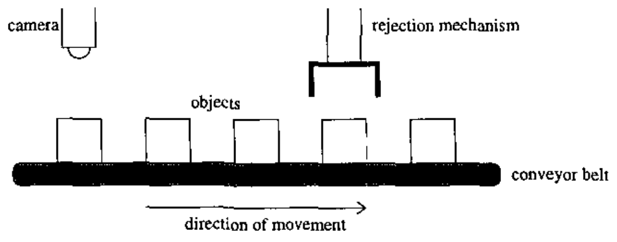

Figure 1: Automated visual inspection.

It must be guaranteed that inspection keeps up with the speed of the conveyor belt. This implies timing constraints for certain parts of the inspection mechanism. The time interval between two consecutive object arrivals provides an upper bound for the processing time of one image, if consecutive objects are handled by the same processor. The arrivals may be strictly periodic or only bounded by a maximum rate.

The rejection mechanism may only be activated after a given amount of time has elapsed once the corresponding image is processed. This introduces constraints on the delay between different tasks.

2.2.3 Sensing

In industrial systems, data of the process under control is acquired by means of sensors [11]. These sensors are scanned, sampled and digitized at rates as fast as 10 times per second. Even higher rates may occur in the sensing of certain physical or chemical processes, such as those in nuclear power plants.

Periodic scanning is one way of performing measurements. This results in a periodic use of the communication media that are used for the transport of the measured values to the controlling processes. Effectively, this technique is equivalent to event handling in a polling strategy. Often, it is sufficient to pass these measured values only if they are significantly different (for example more than 5 percent) from the values measured previously. This results in a reduction of the communication capacity requirements. To reduce the communication requirements even more, it is possible to specify a minimal time delay between consecutive communications of sensor measurements. This prevents measurements occuring at an unnecessarily high rate.

The measured values are used by the controlling process to determine a control policy. The actual control takes place in periodic loops.

2.3 Systems control

Apart from the control of flexible manufacturing systems, many timing constraints can be encountered in more specific control systems. The control of a photocopier is the subject of Section 2.3.1 and aircraft control is discussed in Section 2.3.2.

2.3.1 Photocopier Control

A particular high-volume photocopying machine operates as follows. Latent images are produced on a photo-conductor belt. This belt is welded at one location and so no images may be made across the weld. Therefore, the weld is marked by a master hole in the belt. The time at which the master hole passes a particular point in each full revolution of the belt is recorded by a sensor. The belt is subdivided into image planes which correspond to a single copy. The belt is first electrically charged, after which exposure of the original document decharges the white parts of the original. The image is then developed with toner that sticks to the charged areas of the belt. This image is transferred at a high temperature and under high pressure to a sheet of paper via a second belt. Blank sheets are stored in paper trays. When a sheet is required, it is first separated from a tray. Then, it halts at a stopper. When this stopper is lifted, the sheet is processed without halting. After an image has been transferred to the sheet, it is transported to an output tray. Originals are automatically transported from a tray to the glass plate and returned to the tray after exposure. For the exposure, a capacitor must be charged which takes a long time.

Different types of timing relations exist in this copier. The strictest relations concern synchronization of the different parts of the copier. An image of an original must be accurately positioned on a sheet of paper. This means that the time of the exposure of the original, the time at which the stopper is lifted and the arrival time of an image plane at the appropriate position must be synchronized to within a few milliseconds.

A second type of timing relation stems from sensing the master hole. The master hole sensor is polled during the period that it may pass the sensor. All the operations in the copier control system are synchronized with the moment at which the sensor detects the master hole. Since small variations may occur in the velocity of the photo-conductor belt, the operations in the schedule must be synchronized with each passage of the master hole. This is done by synchronizing the clock that controls the schedule to each passage of the master hole.

Precedence constraints are caused, for example, by the requirements that the flash capacitor is loaded before exposure, that a sheet is separated before the stopper is lifted and that an original is transported to the glass plate before exposure.

2.3.2 Aircraft control

An aircraft control system can be decomposed into a number of application processes. These processes are executed in a periodic way: each application process is dispatched at a specific point relative to the start of an overall system cycle or loop.

Start and completion of different processes and their associated I/O behaviors have to be synchronized such that overruns in the system do not occur [3].

An aircraft control system also contains sensors and actuators. Output to the actuators must be generated with specified frequency, i.e. the task (process) that controls an actuator should be executed periodically. Transport delay — the delay between the reading of sensors and the generation of output of the actuators based upon those readings — must be kept below specified limits.

To fulfill these requirements, an iteration rate is specified for each task. The scheduling strategy must guarantee that the processing of each iteration of the task will be completed within a 'time frame' that is determined by the start time of the iteration and the start time of the next iteration. It does not matter when the processing is performed, provided that it is completed by the end of the frame. The iteration rates required by different tasks differ, but they can be adjusted somewhat to simplify the scheduling. The time needed to execute an iteration of a task is highly predictable, so it can be assumed that these are given [18]. Furthermore, precedence relations between tasks also occur.

3 Timing and Precedence Constraints

In this section, the different timing and precedence constraints are examined for a number of the applications of the previous section. It is argued that Timed Precedence Constraints (TPCs) can describe all precedence and timing constraints in an application. A TPC is a generalization of a normal precedence relation. A TPC specifies a relation between the times at which two *operations or statements* are executed. The TPC $O_1 \xrightarrow{s_0,l_0} O_2$ specifies that operation O_2 must be executed between s_0 and l_0 time units after operation O_1.

Normal precedence relations between operations are present in all applications mentioned. These precedences represent, for example, sequential execution prescribed by a sequential program or a producer-consumer relation between two operations. These precedence relations can be represented by TPCs of the form $O_1 \xrightarrow{c_{max},\infty} O_2$ where c_{max} is the worst-case execution time of operation O_1. However, if worst case execution times are used, then it is not possible to express that the second operation immediately has to follow the first operation. Reason is that the first operation can finish its execution before its (estimated) worst case execution time.

In several of the applications mentioned, certain operations must follow a previous one after a specified period of time. For example, in the TV control the sound is muted when a new channel is selected and the sound is turned back on after 100 ms. This could be specified by a TPC $O_1 \xrightarrow{100,110} O_2$, where l_0 has been chosen somewhat arbitrarily. In the copier control system, many operations must be executed at a fixed time after some other operation within narrow tolerances. This is naturally expressed by TPCs.

TPCs can also be used to express periodicity requirements. Periodicity arises both from periodicity in the application and from the static scheduling approach. Tasks that handle aperiodic events like interrupts must be translated into periodic tasks when static scheduling is used. A schedule is calculated off-line for a certain interval $[0, L]$, where L is the least common multiple of the individual periods of all periodic tasks. This schedule is then executed periodically.

In the existing literature on hard real-time scheduling, tasks are characterized by a period p, a worst-case execution time c, an earliest start time e and a deadline d. Both e and d are relative to the beginning of a period. Without loss of generality, the earliest start time can be taken equal to 0 in the following discussion.

Interrupt-driven tasks must be incorporated in this schedule in such a way that the deadline D on the reaction to the interrupt is always met, i.e. the time between the interrupt and the completion of the result should be less than D. For this, a task is introduced which executes periodically with period p and deadline d. At each execution of this task, it is checked if an interrupt has occurred. If this happens, the interrupt-related task is executed. Otherwise the processor will be left idle.

If no TPCs are used, a period and deadline have to be specified statically. If one assumes that an occurrence of an interrupt just after the scheduled start time of the task is not handled until the next period, p and d must satisfy:

$$p + d \leq D \tag{1}$$

The choice of d determines the length of the period and the strictness of the constraint on the start time.

If d is chosen equal to p, the scheduler can choose the start time within a period n arbitrarily between 0 and $p - c$ (figure 2(a)). Here, c is the worst-case execution time of the interrupt-driven task. This choice allows maximal freedom to the scheduler. However, according to expression (1), the period is now required to be (at most) $D/2$ and the load on the processor due to the task (which is given by the execution time divided by the available time in one period) equals $2c/D$.

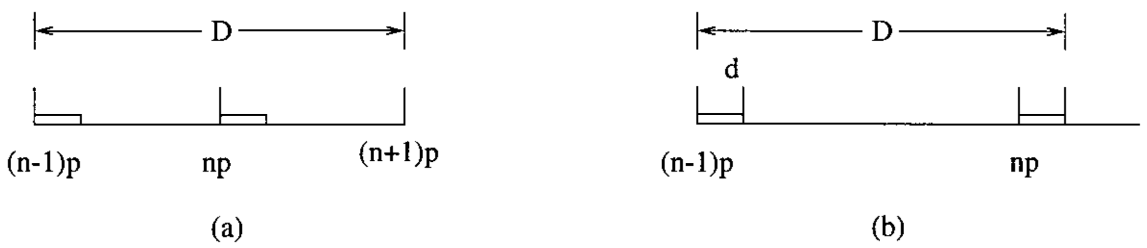

Figure 2: Two possible assignment of periods for TPCs: (a) $d = p, p = D/2$; (b) $d = c, p = D - c$.

On the other hand, the task can be made strictly periodic within $[0, L]$ if d is set equal to c (figure 2(b)). In this case, the scheduler has no freedom to choose a start time. The period of the task can now be reduced to $p = D - c$, according to expression (1). The load on the processor is now reduced to $c/(D - c)$. However, this specification is unnecessarily strict since there is no reason (other than efficiency) why a response should not be given earlier.

When using TPCs, the timing constraints of interrupt-driven tasks can be specified with a TPC $\overset{c,D-c}{\longrightarrow}$ between subsequent activations instead. This allows both freedom for the scheduler and a relatively low average processor utilization. The reason for this is that no (static) choice for d has to be made, whereas the TPC allows a choice of any interval between different activations that satisfies the deadline D.

For a comparison between periodicity induced by a static choice and periodicity induced by TPCs, consider a static specification and a specification with a TPC, where both specifications allow the scheduler intervals of equal length in which an activation must be started. The static specification of the scheduling problem with deadline d allows an interval $[0, d - c)$ for the start time within the period. The utilization is given by $c/(D - d)$, if the period is chosen maximal according to expression (1), i.e. $p = D - d$.

The corresponding TPC is $\overset{D-d,D-c}{\longrightarrow}$. This TPC also specifies an interval of length $d - c$ for the choice of the start time. When the assignments of start times are evenly spread over this interval in the entire scheduling period $[0, L]$, the average utilization due to this TPC equals $c/\frac{1}{2}(l_0 + s_0)) = c/(D - \frac{1}{2}c - \frac{1}{2}d)$. If $c \leq d$, then this average load is smaller than the load that results from the static specification. Likewise, when the TPC is chosen such that the average load equals the load that results from the static specification, the timing constraint due to the TPC is less strict.

In general, very few applications seem to require strictly periodic tasks. Most tasks can be naturally specified by TPCs with $s_0 \neq l_0$ instead. The exception to this are tasks that involve synchronization. Examples of this are the task that awaits the master hole in the copier of section 2.3.1 and the task that awaits the decoding information in the example of the television. These tasks appear strictly periodic in the schedule. However, notice that the clock that drives the schedule, must be reset at each activation of these tasks, otherwise the synchronization between the tasks in the schedule, that are time related to the synchronization point, and the synchronizing signal is lost. This also implies that a static schedule is practically only possible if there is only one synchronizing signal. If more synchronizing signals are present, they must define exactly the same external clock.

4 The DEDOS scheduling model

In this section a scheduling model for the DEDOS environment is presented, in which the aspects of timeliness, reliability and hard real-time scheduling are incorporated. In principle, this model can be extended for soft real-time scheduling. For this purpose, the notation can be easily extended with value functions.

An earlier version of this model is presented in [16]. In contrast with that model, replication is part of the current model, apart from other differences in the level of detail.

The model presented in this paper will be used in the development of scheduling algorithms for the DEDOS design environment.

All the examples in Section 2 have a common characteristic: the applications consist of both hardware and software components; the software is executed on the hardware. In general, the hardware does not consist of a single processor. Instead, a distributed environment is used with a number of possibly inhomogenous processing elements.

In correspondence to this observation, DEDOS applications can be split into two clearly distinguishable parts. Section 4.1 describes a general hardware platform on which an application is executed. In Section 4.2 the description of the software of an application is specified. The general scheduling problem is formulated in Section 4.3.

4.1 Hardware environment

The hardware used in the DEDOS development environment is based on the Eindhoven MultiProcessor System (EMPS)[13]. An EMPS system consists of a number of interconnected crates. A crate contains a number of processors, devices and memory modules which are connected by a common bus. This structure is quite general and can be used for many other systems, too.

In the following definition, *PR*, *DV*, *MM* and *BUS* are distinct collections of processors, devices, memory modules and buses respectively. Processors contain local memories, in which program code and small amounts of data can be stored.

Definition 4.1 (Crates) The crates are described by a 5-tuple $(CR, CRPR, CRDV, CRMM, CRBUS)$, where

- CR is a set of crates,
- $CRPR(cr) \in \mathcal{P}(PR)$ is the set of processors in crate cr, $cr \in CR$,

- $CRDV(cr) \in \mathcal{P}(DV)$ is the set of devices in crate cr, $cr \in CR$,
- $CRMM(cr) \in \mathcal{P}(MM)$ is the set of memory modules in crate cr, $cr \in CR$, and
- $CRBUS(cr) \in BUS$ is the common bus in crate cr, $cr \in CR$.

$\square$

For convenience, we introduce four auxiliary functions.

Definition 4.2 (Auxiliary functions)

- $PRCR(pr)$ is the (unique) crate in which processor pr is situated, $pr \in PR$,
- $DVCR(dv)$ is the (unique) crate in which device dv is situated, $dv \in DV$,
- $MMCR(mm)$ is the (unique) crate in which memory module mm is situated, $mm \in MM$, and
- $BUSCR(bus)$ is the (unique) crate in which bus bus is situated, $bus \in BUS$.

$\square$

To make this a valid definition, it is assumed that $CRPR(cr_1) \cap CRPR(cr_2) = \emptyset$, $CRDV(cr_1) \cap CRPR(cr_2) = \emptyset$, $CRMM(cr_1) \cap CRMM(cr_2) = \emptyset$, and $CRBUS(cr_1) \neq CRBUS(cr_2)$for $cr_1, cr_2 \in CR$ with $cr_1 \neq cr_2$.

Some of the resources have a certain capacity. For example, if a piece of code, that consists of i instructions, is executed on a processor with a capacity of m MIPS, it is finished after i/m microseconds. If a piece of data of d bits is communicated via a bus with capacity c' bits per second, then this communication is finished after d/c' seconds. The capacity of a device is defined likewise. The capacity of a memory module gives the number of bits a memory module can contain.

Definition 4.3 (Capacity)

- $PRCAP(pr)$ is the capacity of processor pr, $pr \in PR$,
- $DVCAP(dv)$ is the capacity of device dv, $dv \in DV$,
- $MMCAP(mm)$ is the capacity of memory module mm, $mm \in MM$,
- $BUSCAP(bus)$ is the capacity of bus bus, $bus \in BUS$.

$\square$

Because devices are not uniform in general, we must distinguish different device types. Examples of devices are sensors, actuators and large storage media such as disks.

Definition 4.4 (Device types) $DVT(dv) \in DVTS$ is the type of device dv. Here $DVTS$ is the collection of device types. $\square$

In general, the hardware used in the DEDOS environment consists of a network of interconnected crates. The crates are interconnected by dedicated interprocessor links. Links can also connect two processors in the same crate. Like other resources, links also have a predefined capacity.

Definition 4.5 (Links) *LI* is the set of dedicated interprocessor links. $LICAP(li)$ is the capacity of link li, $li \in LI$. $LIPR(li)$ is the set of (two) processors that are connected by link li. □

For convenience, we introduce the concept of communication media, which consist of buses and dedicated links.

Definition 4.6 (Communication media) $CM = BUS \cup LI$ is the set of communication media. Here, we assume $BUS \cap LI = \emptyset$. Which resources are connected by a certain communication medium is given by function $CMCON : CM \rightarrow \mathcal{P}(PR \cup DV \cup MM)$, which is defined by

$$CMCON(cm) = \begin{cases} \{pr \in PR \mid PRCR(pr) = BUSCR(cm)\} \cup & \\ \{dv \in DV \mid DVCR(dv) = BUSCR(cm)\} \cup & \text{if } cm \in BUS \\ \{mm \in MM \mid MMCR(mm) = BUSCR(cm)\} & \\ LIPR(cm) & \text{if } cm \in LI \end{cases}$$

The capacity of a communication medium is given by

$$CMCAP(cm) = \begin{cases} BUSCAP(cm) & \text{if } cm \in BUS \\ LICAP(cm) & \text{if } cm \in LI \end{cases}$$

□

With respect to the connectivity of the crate network, it is assumed that there exists a communication path between each pair of processors, i.e.

$$(\forall pr_1, pr_2 \in PR :: path(pr_1, pr_2)),$$

In this definition, $path : PR \times PR \rightarrow \mathbb{B}$ is defined by

$$\begin{aligned} path(pr_i, pr_i) &\equiv true \\ path(pr_i, pr_j) &\equiv (\exists pr_k \in PR, cm \in CM :: \{pr_i, pr_k\} \subseteq CMCON(cm) \wedge path(pr_k, pr_j)) \end{aligned}$$

Figure 3 reflects an example of a specific static structure. Two crates with processors, devices, buses, memory modules and dedicated links are shown.

Processors, devices, memory modules, buses and links are all the resources in a DEDOS system.

Definition 4.7 (Resources) The set of resources is given by $RS = PR \cup DV \cup MM \cup BUS \cup LI$. □

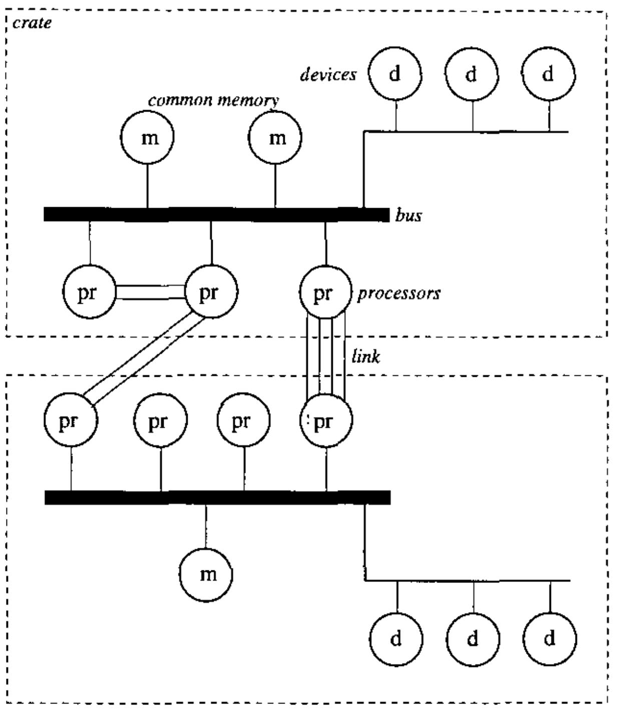

Figure 3: DEDOS static structure.

4.2 Dynamic structure

In the DEDOS environment, a real-time application is described in an object-oriented language, called DEAL (DEDOS Application Language) [8][14]. The object-oriented development paradigm is chosen for its reusability and extendibility. DEAL provides transparency to the user from hardware aspects such as distribution across the network and communication.

DEAL is based on the C++ language because the language is widely used. The language is extended with concurrency, atomicity, remote procedure calls and timing annotations. The user can specify timing relations between different program entities by these timing annotations.

There are two kinds of timing annotations. A time measurement expresses that a statement happens to be executed at a certain moment. The execution of other statements can be related to this moment. For example, the time measurement $S_1[?t]$ expresses that statement S_1 supplies an execution moment denoted by t. This time variable t can be used to express a timing relation with other statements. Therefore, timing requirements are introduced. An example of this is $S_2[\geq t+2, \leq t+4]$, which expresses that statement S_2 should start its execution between 2 and 4 time units after the time that is given by time variable t; in this context, t represents the time at which statement S_1 is executed. The program domain and the timing domain are independent, i.e. timing variables do not occur in the program code and programming variables do not occur in timing annotations. Notice that a time measurement and a timing requirement directly define a TPC.

A preprocessor transforms a DEAL application in pure C++ code by translating DEAL specific constructs into C++ constructs. During the preprocessing also timing information is produced, which is deduced from the timing annotations. The resulting timed precedence constraints are input to the scheduling stage.

4.2.1 Objects

An object is a piece of data, together with methods and activities to inspect or modify this data. The data can only be accessed through its corresponding methods and activities.

Definition 4.8 (Objects) The collection of objects is given by set OB. □

A method consists of program code, which describes how the data is manipulated. Within the code of a method, a call to a method of the same or a different object is allowed.

An activity is a special kind of method. It is executed upon creation of the corresponding object. While ordinary methods are executed once per call, an activity is executed periodically. In general, the execution of an activity results in a series of method calls. Such a sequence of method calls is termed a thread. The calls in a thread are not restricted to one object: a thread may cross object boundaries. In the scheduling stage, a start time has to be determined for each activity.

4.2.2 Beads

It is clear that the code of methods and activities has to be executed on processors. In general, an entire method or activity is too large to be executed as one whole. Therefore preemption is allowed, i.e. the execution of a piece of code may be interrupted and resumed at a later time. In the meantime, another piece of code may be executed on the same processor. In this way a higher processor utilization is achieved.

Each time the processor switches to the execution of a different piece of code, a context switch is made. If full preemption is allowed, i.e. if the execution can be interrupted at any moment, then this could lead to too many context switches. No preemption would lead to too much inflexibility for the scheduling stage. Therefore semi-preemptability is allowed: code in execution can be interrupted at given predefined moments, the so-called preemption points.

The preemption points are placed in logical places. These are for example places where a continuation of execution is not immediately possible, due to device accesses or method calls, which cannot be dealt with in a negligible amount of time. Other preemption points are introduced as a result of timing annotation in the source code, and at places where the execution of program code starts or ends.

Definition 4.9 (Preemption points) A preemption point indicates at which moment the execution of a piece of code may be started, ended, interrupted or resumed, i.e. when a context switch is allowed. □

It is assumed that the creation of preemption points is dealt with in the preprocessing stage; the preemption points are therefore considered given.

Pieces of code between consecutive preemption points should be executed atomically; for example, no communication takes place within such a piece of code. Therefore, the concept of beads is introduced. A bead is the unit of non-preemption and scheduling. It is assumed that the collection of beads is given.

Definition 4.10 (Beads) A bead is the non-preemptable part between a preemption point and all its successor preemption points. A preemption point pp_2 is a successor of a preemption point pp_1 iff an execution exists in which there are no other preemption points between pp_1 and pp_2, and pp_1 and pp_2 are located in the same method or activity. Set B is the collection of all beads in the given application. Function $BOB : B \rightarrow OB$ expresses to which object a bead belongs. □

The execution of the entire system consists of the execution of beads on processors. So, it has to be determined which bead is executed on which processor.

Definition 4.11 (Bead assignment) $BA(b) \in PR$ is the processor on which bead $b \in B$ is executed. □

To reduce the amount of communication, object bead constraints are introduced, which express that beads, belonging to the same object, should be executed on the same processor.

Definition 4.12 (Object bead constraints) The object bead constraints are given by

$$BA(b_1) = BA(b_2)$$

for all $b_1, b_2 \in B$ with $BOB(b_1) = BOB(b_2)$ □

As a result of the object bead constraints, each object is assigned to a single processor, which is reflected in the following definition.

Definition 4.13 (Object assignment) $OBA(ob) \in PR$ is the (unique) processor to which beads of object $ob \in OB$ are assigned, i.e.

$$OBA(ob) = BA(b)$$

for all $ob \in OB$ and some $b \in B$ with $BOB(b) = ob$. □

In the preprocessing stage, timing information is extracted from the given application. Part of this information is the identification of beads and an estimate of their processing requirements. For example, if bead b involves i instructions, and if it is executed on processor pr with capacity $m = PRCAP(pr)$ MIPS, then the execution of bead b on processor pr would take i/m microseconds. Only a worst case estimate of the processor requirement is calculated.

In the scheduling stage, it has to be determined for each bead at which time it should start its execution. Given the start time and the processing requirement of a bead and given its corresponding bead assignment, it is possible to derive the time at which the execution of that bead is finished.

Definition 4.14 (Bead timing) $BPR(b)$ is the processor requirement of bead $b \in B$. $BS(b)$ is the start time of bead $b \in B$. The end time of bead $b \in B$ is denoted by $BE(b)$ and is defined by $BE(b) = BS(b) + r/c$, where $r = BPR(b)$ is the processing requirement of b, and $c = PRCAP(BA(b))$ is the capacity of the processor on which b is executed. □

Note that in the scheduling stage, function BS has to be determined, and that the value of function BE depends on the chosen bead assignment (and start times).

4.2.3 Bead timing relations

In Section 3, it is shown in general terms how timing relations can be expressed by TPCs. This section focusses on the determination of TPCs in terms of beads. It is assumed that the set of TPCs is determined in the preprocessing stage: based on the collection of beads and the program code with the corresponding timing information, it is possible to derive timing relations between different beads. This is illustrated by means of an example, which is given in figure 4. For reasons of clarity, the example is not coded in DEAL, but in an imaginary object-oriented language.

```
OBJECT read_sensors
METHOD read(in nr: Int; out v: Real)
BEGIN
    $\rhd^1$
    IF nr = 1
    THEN $\lhd^2$ sensor1.read(v) $\rhd^3$
    ELSE
      IF nr = 2
      THEN $\lhd^4$ sensor2.read(v) $\rhd^5$
      ELSE $\lhd^6$ error.write() $\rhd^7$
    $\lhd^8$ END
OBJECT sensor1
DATA temperature: Real
METHOD read(out t: Real)
BEGIN
    $\rhd^9$ t := temperature $\lhd^{10}$
END
OBJECT sensor2
DATA distance: Real
METHOD read(out d: Real)
BEGIN
    $\rhd^{11}$ d := distance $\lhd^{12}$
END
OBJECT error
METHOD write()
BEGIN
    $\rhd^{13}$ ... $\lhd^{14}$
END
```

Figure 4: Example program.

In this example, object **read_sensors** provides services to the environment to read two sensors, one for reading a temperature (object **sensor1**) and one for reading a distance (object **sensor2**). The numbered $\rhd$ and $\lhd$ symbols denote the preemption points. The $\rhd$ symbols correspond to points were (parameter) data is possibly received, whereas $\lhd$ symbols denote the possible transmission of data.

According to the definition, five beads can be identified (see figure 5). The first, bead b_1, is bounded by preemption points 1, 2, 4 and 6; this bead corresponds to the check which sensor has to be read. The second bead (b_2) is bounded by preemption points 9 and 10 and models the actual reading of the thermometer. The third bead (b_3), bounded by preemption points 11 and 12, corresponds to the reading of the distance sensor. The fourth bead (b_4) between preemption points 13 and 14 represents the call to the error handling object. Finally, the fifth bead (b_5) is added to represent the correct ending of the read-method in object read_sensors.

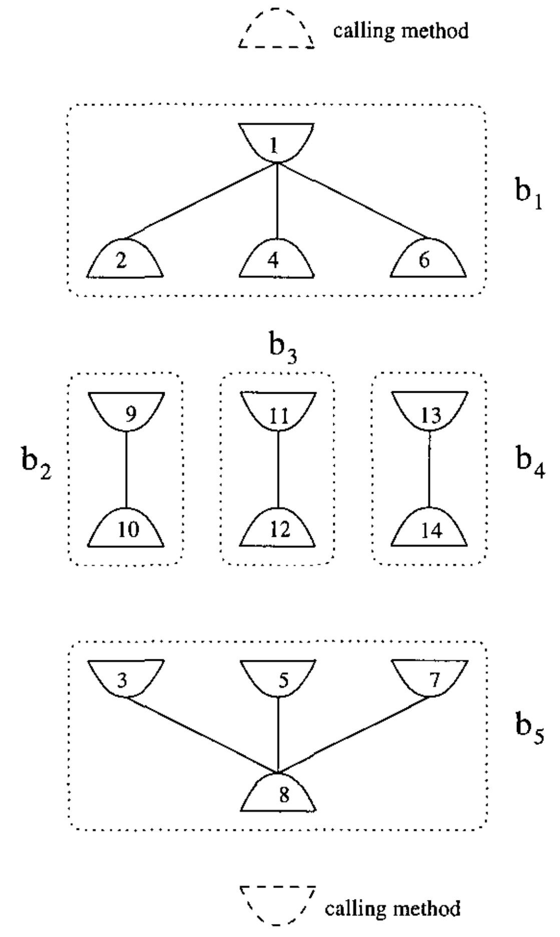

Figure 5: Beads in the example program.

It is clear, that if the input specifies that a temperature has to be read, the actual reading of the thermometer has to follow the validation of this input. Therefore, bead b_2 should be preceeded by bead b_1, i.e. the execution of b_1 should be finished before the execution of bead b_2 is started ($BE(b_1) \leq BS(b_2)$). The same goes for beads b_3 and b_1: b_1 should precede b_3, because bead b_3 corresponds to the execution of the method, that is called at the end of bead b_1. Bead b_4 should succeed bead b_1, due to the same reason. Finally, bead b_5 should succeed beads b_2, b_3 and b_4, because this bead corresponds to the end of the method. These timing relations are termed precedence constraints, and should be taken into account in the scheduling stage. Figure 6 gives a visual representation of the precedence constraints between the beads.

Precedence constraints are just one type of timing constraints. In general, more complex

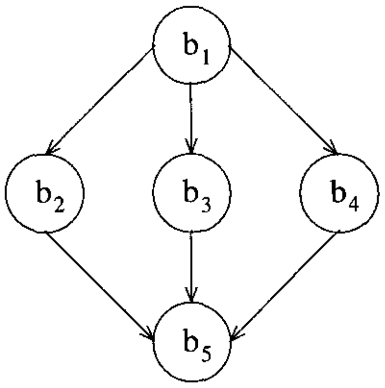

Figure 6: Beads and precedence constraints from the example program.

timing relations can be expressed by TPCs. It is assumed that all timing relations between beads are expressed by these TPCs (see also Section 3).

Definition 4.15 (Timed precedence constraints) The timed precedence constraints are given by

$$BS(b_1) + s \leq BS(b_2) \leq BS(b_1) + l$$

for all $(b_1, s, l, b_2) \in TPC$, where TPC is a given set of timed precedence constraints. □

A timing relation induced by a timed precedence constraint is not symmetrical (i.e. $(b_1, s, l, b_2) \neq (b_2, s, l, b_1)$). It is assumed that between each pair of beads, at most one timed precedence constraint can be present. It is easy to see that more than one timed precedence constraint on the same two beads can be transformed into a single constraint (at least: if the constraints are not inconsistent). For example, TPCs $(b_1, 2, 6, b_2)$ and $(b_1, 4, 8, b_2)$ have the same effect on the schedule as the single TPC $(b_1, 4, 6, b_2)$.

A (normal) precedence constraint is easily expressed by a timed precedence constraints. For example, suppose there is a precedence constraint between beads b_1 and b_2, i.e. $BE(b_1) \leq BS(b_2)$ should be met. This is achieved by adding tuple $(b_1, r/s, \infty, b_2)$ to the set of timed precedence constraints TPC, where $r = BPR(b_1)$ is the processor requirement of bead b_1, and $s = PRCAP(BA(b_1))$ is the capacity of the processor on which bead b_1 is executed.

4.2.4 Bead communication

Sometimes data is sent from one bead to a different bead. For example, if in a bead a method is called in a different object, then in general parameter data should be transferred at the method call. The results should be transferred back to the calling bead at the end of the execution of the method.

Definition 4.16 (Communicating beads) Function $BCM : B \times B \rightarrow \mathbb{B}$ expresses which beads communicate data. $BDS(b_1, b_2)$ is the amount of data that is exchanged between beads b_1 and b_2. If beads b_1 and b_2 do not communicate data (i.e. $\neg BCM(b_1, b_2)$ holds), then $BDS(b_1, b_2)$ is not defined. □

Note that function BCM is not symmetrical in general (i.e. $BCM(b_1, b_2)$ does not necessarily equal $BCM(b_2, b_1)$), since communication is characterized by a direction: data is communicated from one bead to an other bead.

It is assumed that both functions *BCM* and *BDS* are determined in the preprocessing stage before scheduling. From the definition of preemption points it follows that there is at most one communication between each pair of beads.

4.2.5 Device management

Devices are used or managed by objects. Each device is managed by one object, i.e. all accesses of a device occur via the managing object. Therefore, there is a relation between devices and objects. In general, this relation is prescribed by the application. For example, a sensor and an object that controls this sensor are clearly related. So, it is assumed that this relation is predefined, i.e. the relation is known before the scheduling stage.

Definition 4.17 (Device assignment) Function $DVOB : DV \rightarrow OB$ expresses for each device by which object it is managed. □

Note that this definition allows that multiple devices are managed by one object. However, a device can only be managed by a single object.

Because a device can only be accessed through its managing object, this device and this object should be located in the same crate. This is expressed by the device constraints.

Definition 4.18 (Device constraints) The device constraints are given by

$$PRCR(OBA(ob)) = DVCR(dv)$$

for all objects $ob \in OB$ and devices $dv \in DV$ with $DVOB(dv) = ob$. □

Device accesses resemble method calls, because in both cases data is sent from one resource (a processor) to a different resource (a processor or device). The receiving resource is occupied for some time, after which data is returned to the first resource.

Therefore, device accesses are treated in a similar way as method calls. A method call is represented by a series of beads: first a bead of the calling object, followed by one or more beads of the called method, followed by again a bead of the calling object. A device access is modeled by a bead of the object, which accesses the device, followed by a bead, which represents the time that the device needs for the actual access, followed again by a bead of the accessing object. The bead that corresponds to the actual device access is termed a device bead and is part of the scheduling problem.

It is assumed that the set of beads B also contains these device beads. To distinguish device beads from processor beads, a function is introduced which gives the bead type.

Definition 4.19 (Bead types) Function $BT : B \rightarrow \{p, d\}$ gives the type of a bead. $BT(b) = p$ iff bead $b \in B$ is a processor bead; $BT(b) = d$ iff bead b is a device bead. □

To realize the communication between objects and devices, functions *BCM* and *BDS* are also defined on device beads. For example, if a bead b_1 ends with a device access, which corresponds to the execution of a device bead b_2, then $BCM(b_1, b_2)$ holds, and the data that is sent from b_1 to b_2 has size $BDS(b_1, b_2)$. If the data resulting from this device access is returned to bead b_3, then also $BCM(b_2, b_3)$ holds and this data has size $BDS(b_2, b_3)$.

Furthermore, bead timing functions BS (bead start time) and BE (bead end time) are defined on device beads, too. The device capacity needed by a device bead is given by function BPR (bead processing requirement).

For device beads, the range of bead assignment function BA is extended in order to express to which device a device bead belongs, i.e. function BA has range $PR \cup DV$. In the scheduling stage, constraints with respect to bead types have to be met.

Definition 4.20 (Bead type constraints) The bead type constraints are given by

$$BA(b) \in PR \Leftrightarrow BT(b) = p$$

for all beads $b \in B$, and

$$BA(b) \in DV \Leftrightarrow BT(b) = d$$

for all beads $b \in B$. □

4.2.6 Execution graph

Given the collection of beads and the time and communication dependencies between beads, it is possible to model software applications by means of an execution graph. The nodes in this directed graph correspond to beads, while the arcs represent both communications and timed precedence constraints. The execution graph is the input for the scheduling algorithm; it has to be mapped to the given hardware architecture (after some transformations, which are discussed in the next sections), and start times have to be provided for the beads.

Definition 4.21 (Execution graph) The execution graph is given by a directed graph $G = (V, A)$, where

- $V = B$, and
- $A = A_{TPC} \cup A_{CM}$.

Here, A_{TPC} is the set of arcs induced by the timed precedence constraints, i.e. $A_{TPC} = \{(b_1, b_2) | \exists s, l \in \mathbb{N} :: (b_1, s, l, b_2) \in TPC\}$, and A_{CM} is the set of arcs induced by the communication behavior, i.e. $A_{CM} = \{(b_1, b_2) | BCM(b_1, b_2)\}$. □

In fact, this graph can be considered a labeled graph. A distinction is made between arcs that correspond to timed precedence constraints and arcs that correspond to communications.

Definition 4.22 (Label function) The label of arc $(b_1, b_2) \in A$ of graph $G = (V, A)$ is given by

$$L((b_1, b_2)) = \begin{cases} (s, l) \text{ with } (b_1, s, l, b_2) \in TPC & \text{if } (b_1, b_2) \in A_{TPC} \\ BDS(b_1, b_2) & \text{if } (b_1, b_2) \in A_{CM}. \end{cases}$$

□

The case that beads both communicate and have a TPC is discussed in Sections 4.2.8 and 4.2.9. Effectivily, a communication between beads is replaced by a number of new beads and TPCs.

4.2.7 Replication

For the HRT part of DEDOS applications, reliability is achieved by the introduction of replication. For example, if a processor is malfunctioning, the program code it is executing is not valid anymore. To guarantee the correct continuation of the execution of the system, a copy of the program code is also executed on one or more other processors. In case of a malfunctioning processor, control is given to the processor(s) on which the replicated code is executing.

The unit of code replication is the object. Depending on the fault hypothesis, a number of replicated objects are introduced for each original object. The number of replicas is determined in a preprocessing stage. For the moment, it is assumed that the result of this stage is known.

Definition 4.23 (Number of replicas) The number of replicas is given by function NR; $NR(ob) \in \mathbb{N}$ is the number of replicas of object $ob \in OB$. □

The introduction of replicated objects results in an extension of the set of objects with replicas. A function is introduced which expresses if objects are replicas.

Definition 4.24 (Replicated objects) The collection of objects including replicas is given by set OB'. Function $RPOB : OB' \times OB' \rightarrow \mathbb{B}$ expresses which objects are replicas. $RPOB(ob_1, ob_2)$ holds iff ob_1 and ob_2 are replicas. □

It is clear that the addition of replicated objects results in additional beads. Each bead of the original object is present in each replicated object. Therefore, the set of beads is extended; the result of this extension is bead set B'.

Not only additional beads, but also extra TPCs are introduced. For example, if there is a TPC between beads b_1 and b_2 with parameters s and l, and if both beads are replicated, then there is a TPC from each replica of bead b_1 to each replica of bead b_2 with values s and l. The extended set of TPCs is denoted by TPC'.

The way in which the communication behavior is influenced by the introduction of replicated beads and objects depends on the communication protocol used. A reliable communication mechanism based on replicated mailboxes is discussed in [19]. Because a discussion of such communication protocols is not the subject of this paper, it is assumed that it is known how the communication between replicated beads is established, i.e. functions BCM and BDS are suitably extended into functions BCM' and BDS' respectively.

The replication of beads and timed precedence constraints and the adapted communication structure result in an extension of the execution graph.

Definition 4.25 (Execution graph extension for replication) The extended execution graph for replication is given by the directed graph $G' = (V', A')$, where

- $V' = B'$, and
- $A' = A'_{TPC} \cup A'_{CM}$.

Here, A'_{TPC} is the set of arcs induced by the timed precedence constraints, i.e. $A'_{TPC} = \{(b_1, b_2) | \exists s, l \in \mathbb{N} :: (b_1, s, l, b_2) \in TPC'\}$, and A'_{CM} is the set of arcs induced by the communication behavior, i.e. $A_{CM} = \{(b_1, b_2) | BCM'(b_1, b_2)\}$. □

The label function is extended likewise.

Definition 4.26 (Extended label function) The label of arc $(b_1, b_2) \in A'$ of the extended graph $G' = (V', A')$ is given by

$$L'((b_1,b_2)) = \begin{cases} (s,l)\text{with } (b_1,s,l,b_2) \in TPC' & \text{if } (b_1,b_2) \in A'_{TPC} \\ BDS'(b_1,b_2) & \text{if } (b_1,b_2) \in A'_{CM}. \end{cases}$$

□

All other functions and constraints defined on beads and objects, are suitably extended to the domains B' and OB' respectively. These extended functions are denoted by the addition of primes.

It is clear that the replication of objects introduces constraints on the assignment of beads to processors, i.e. two beads that belong to two replicated objects, should be assigned to different processors.

Definition 4.27 (Bead replication constraints) The bead replication constraints are given by

$$BA'(b_1) \neq BA'(b_2)$$

for all $b_1, b_2 \in B'$ with $RPOB(BOB'(b_1), BOB'(b_2))$ □

4.2.8 Communication beads

If two communicating beads are assigned to different processors, it is possible that these processors are not connected directly. During scheduling, means have to be provided to make this communication possible, i.e. a route of resources has to be chosen along which the data is transported and time should be reserved during which actual communication can take place.

Program code has to be provided for each intermediate processor. This additional code supports the receipt and transmission of this data along the communication route.

For example, if in the example of figure 7 route $< pr_1, bus_1, pr_3, link_2, pr_7 >$ is used for the communication of data between a bead on pr_1 and a bead on pr_7, then code has to be provided for processor pr_3 which takes data from bus bus_1 and sends it to link $link_2$. The execution time of this code depends on the size of the data and the capacities of the communication media and processor under consideration.

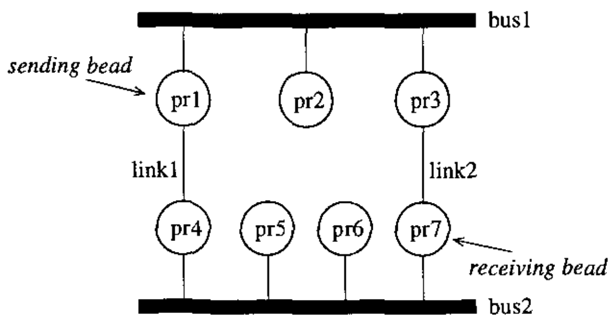

Figure 7: Communicating beads.

It is clear that these additional pieces of code can be considered to be another kind of beads: time has to be reserved on processors for the execution of this extra code.

Not only must extra time be reserved on processors, but also communication resources (links, buses) are occupied in a certain interval of time during a communication (bus_1 and $link_2$ in the example of figure 7). This occupation is modeled by additionial beads. The timing information of these beads expresses the usage and timing of the corresponding communication resources. The type of the extra beads is defined to be c, i.e. $BT(b) = c$. Therefore, the range of bead type function BT is extended to $\{p, d, c\}$, and the range of bead assignment function BA is extended with set CM, i.e. $BA : B \rightarrow PR \cup DV \cup CM$.

In effect, the addition of these two types of beads corresponds to a transformation of the execution graph: each arc reflecting a communication between two beads has to be replaced by a series of beads. These additional beads are used to model the use of resources to achieve the desired communication. The resulting execution graph solely contains arcs corresponding to timed precedence constraints. If suitable timing constraints are added for the additional beads, then it can be guaranteed that actual communication can be established. So, the scheduling algorithm has to find a route of resources for each pair of communicating beads. These beads have to be assigned to the appropriate resources and start times have to be determined for these beads.

The way in which communication beads have to be inserted is illustrated by a continuation of the example of figure 7. Suppose that route $< pr_1, bus_1, pr_3, link_2, pr_7 >$ is used for the communication of data between a bead b on pr_1 and a bead b' on pr_7 (i.e. $BCM(b, b')$ holds), then the following beads have to be added:

- a bead b_1 that corresponds to the program code (on processor pr_1) that puts the data provided by b into bus_1 (this code also puts the data in the right format; for example, it provides message headers and compresses the data),
- a bead b_2 that corresponds to the occupation time of bus_1,
- a bead b_3 that corresponds to the program code for the receipt of the data on processor pr_3,
- a bead b_4 that corresponds to the program code on processor pr_3 that sends the data to $link_2$,
- a bead b_5 that corresponds to the occupation of $link_2$, and
- a bead b_6 on processor pr_7 that corresponds to the program code that receives the data and passes it onto bead b'.

The introduction of these beads leads to the following additional timing constraints:

- b should be finished before b_1 starts, i.e. TPC $(b, \frac{BPR(b)}{PRCAP(pr_1)}, \infty, b_1)$ should be met,
- b_1, b_2 and b_3 should be executed during the same time interval, which corresponds to TPCs $(b_1, 0, 0, b_2)$ and $(b_2, 0, 0, b_3)$,
- b_3 should preceed b_4 (TPC $(b_3, \frac{BPR(b_3)}{PRCAP(pr_3)}, \infty, b_4)$),
- b_4, b_5 and b_6 should be executed at the same time (TPCs $(b_4, 0, 0, b_5)$ and $(b_5, 0, 0, b_6)$), and

- b_6 should preceed b' (TPC $(b_6, \frac{BPR(b_6)}{PRCAP(pr_7)}, \infty, b')$).

If communication takes place between two beads that are assigned to the same processor, it is not necessary to introduce additional communication beads together with the corresponding timing relations; the data can be exchanged between the beads directly.

Definition 4.28 (Local and remote procedure calls) A local procedure call is a method call to a method of an object on the same processor. A remote procedure call is a method call to a method of an object on a different processor. □

In case of a local procedure call, no additional communication beads have to be provided. The addition of communication beads clearly depends on the assignment of beads to processors.

4.2.9 Execution graph transformation

From the previous section it is clear that the realisation of communication effectively comes down to replacing arcs in the execution graph between communicating beads by a series of beads and arcs corresponding to additional beads and timed precedence constraints. This process results in a transformed execution graph.

Definition 4.29 (Transformed execution graph) The transformed execution graph is given by the directed graph $G'' = (V'', A'')$, where

- $V'' = B''$, and
- $A'' = A''_{TPC}$.

Here, B'' is the set of beads extended with the communication beads, TPC'' is the set of timed precedence constraints extended with the TPCs concerning communication beads, and A''_{TPC} is the set of arcs induced by the timed precedence constraints, i.e. $A''_{TPC} = \{(b_1, b_2) | \exists s, l \in \mathbb{N} :: (b_1, s, l, b_2) \in TPC''\}$. □

The label function is transformed likewise.

Definition 4.30 (Transformed label function) The label of arc $(b_1, b_2) \in A''$ of the transformed execution graph $G'' = (V'', A'')$ is given by

$$L''((b_1, b_2)) = (s, l)\text{with } (b_1, s, l, b_2) \in TPC''$$

□

All functions concerning beads and objects are suitably extended for the additional communication beads. They are denoted by two primes; for example, $BPR''(b)$ is the resource requirement of bead $b \in B''$.

4.2.10 Blocks

The execution graphs, resulting from the graph extension for replication and tranformation for communication are possibly very large. With respect to the determination of the timing, the total problem instance may be too large to be solved directly. To reduce the problem size, the scheduling algorithm may cluster beads into blocks. First, the timing behaviour of blocks is determined, followed by the timing behaviour of beads within a block.

Beads with very strict timing constraints are logical candidates to be grouped into one block. For example, if a bead should start its execution directly after an other bead is finished, then these two beads can be regarded as one block. Another example are beads in case of a remote procedure call: three beads corresponding to a communication along a bus or link should be executed at the same time. Therefore it suffices to treat these three beads as one block. The timing behaviour of this block directly defines the timing behaviour of its beads.

Definition 4.31 The blocks are given by a triple $(BL, BLRS, BLB)$, where

- BL is a set of blocks,
- $BLRS(bl) \in \mathcal{P}(RS)$ is the collection of resources used by beads in block bl, $bl \in BL$, and
- $BLB(bl) \in \mathcal{P}(B'')$ is the collection of beads corresponding to block bl, $bl \in BL$.

□

During the scheduling stage, a time interval is determined for each block. During this interval, all resources of that block are reserved for the execution of the beads of that block.

A worst case estimate of the time required for this block is needed. This estimate depends on the object assignment and the capacities of the resources corresponding to that block. A time interval is represented by a start time and a time requirement.

Definition 4.32 (Block timing) $BLS(bl)$ is the start time of the interval in which the communication of block $bl \in BL$ is scheduled. The length of this interval is given by time requirement $BLTR(bl)$. The time interval ends at time $BLE(bl) = BLS(bl) + BLTR(bl)$. □

All the resources of a block are considered occupied during the entire interval, which is determined for that block; these resources cannot be used for other purposes during this time interval. The exact start times of beads are chosen after the determination of the time intervals of blocks.

Improving efficiency by using blocks can also be achieved in other ways. For example, in the execution of the reliable message protocol [1], many short messages are exchanged between a number of processors. There is a small chance that these messages collide. Therefore it is not considered worthwhile to schedule all these communications in detail. This can be modelled by introducing a block that corresponds to all processors and communication media involved in the execution of the protocol. All communications should take place in the time interval that is determined for the corresponding block.

Apart from efficiency, there is another motivation for the use of blocks. If atomicity of blocks is assured, then it is possible to realize consistency of a device. Suppose for example that several activities use the same file on a disk. If the execution of the beads in the different activities are interleaved, consistency of the file may be violated. If in each activity the beads that correspond to file handling are grouped into a block, and if these blocks are executed under mutual exclusion, then the consistency of the file is guaranteed. This view on the use of blocks corresponds to the use of transactions in [7]. To assure mutual exclusion, corresponding constraints are needed.

Definition 4.33 (Mutual exclusion for blocks) The mutual exclusion constraints for blocks are given by

$$BLE(bl_1) < BLS(bl_2) \vee BLE(bl_2) < BLS(bl_1)$$

for all $bl_1, bl_2 \in blme$, where $blme \in \mathcal{P}(BL)$ is a given set of blocks that should be executed under mutual exclusion. □

Blocks can also be used to provide the possibility of hierarchical application design. If only the timing and communication behaviour of a piece of code is known, and the exact program code is not yet known, then it is possible to reserve or schedule resources and time for the execution of this code by introducing a block which involves the needed resources.

4.2.11 Implementation constraints

Processors and devices are considered single purpose resources, i.e. only one processor, communication or device bead can be executed at any time on the corresponding resource. Therefore, the result of the scheduling stage should meet certain no-overlap constraints.

Definition 4.34 (No-overlap constraints) The no-overlap constraints are given by

$$BE''(b_1) < BS''(b_2) \vee BE''(b_2) < BS''(b_1)$$

for all $b_1, b_2 \in B''$ with $BA''(b_1) = BA''(b_2)$ □

Note that the no-overlap constraints implement the non-preemptiveness of beads.

With respect to the assignment of objects to processors, it has to be guaranteed that the number of devices used does not exceed the number of available devices. To this end, an auxiliary function is defined which gives for each crate *cr* and each device type *dvt* the number of available devices of type *dvt* in crate *cr*.

Definition 4.35 (Number of available devices) $NADV(cr, dvt)$ is the number of available devices of type $dvt \in DVTS$ in crate $cr \in CR$, i.e. $NADV(cr, dvt) = (\#dv \in DV \; : \; DVCR(dv) = cr \; : \; DVT(dv) = dvt)$. □

Another auxiliary function is defined, which expresses the number of used devices of a certain type in a crate, given the object assignment.

Definition 4.36 (Number of used devices) Given an object assignment OBA, $NUDV(cr, dvt)$ is the number of used devices of type $dvt \in DVTS$ in crate $cr \in CR$, i.e. $NUDV(cr, dvt) = (\#ob \in OB, dv \in DV \; : \; PRCR(OBA(bob)) = cr \wedge DVOB(dv) = ob \; : \; DVT(dv) = dvt)$. □

Now, it is straightforward to define the constraints that specify that the number of used devices of a certain type may not exceed the number of available devices of that type.

Definition 4.37 (Device type constraints) The device type constraints are given by

$$NUDV(cr, dvt) \leq NADV(cr, dvt)$$

for all crates $cr \in CR$ and device types $dvt \in DVTS$. □

4.3 Scheduling problem

Based on the model introduced in sections 4.1 and 4.2, the entire scheduling problem is defined as follows.

Definition 4.38 (Scheduling problem)
Given

- the hardware part of a DEDOS application (i.e. the collections of crates, processors, devices, memory modules, buses, links, the interconnection structure and the capacities of these resources),
- the software part of a DEDOS application (in the form of an execution graph $G' = (V', A')$, which is extended for replication),

determine

- a suitable tranformation of the execution graph $G'' = (V'', A'')$ for the realization of bead communication,
- the bead assignment $BA'' : OB'' \rightarrow PR$,
- block start times $BLS(bl)$, for all blocks $bl \in BL$
- bead start times $BS''(b)$, for all beads $b \in B''$,

subject to

- the object bead constraints, i.e.
$$BA''(b_1) = BA''(b_2)$$
for all $b_1, b_2 \in B''$ with $BOB''(b_1) = BOB''(b_2)$,
- the timed precedence constraints, i.e.
$$BS''(b_1) + s \leq BS''(b_2) \leq BS''(b_1) + l$$
for all $(b_1, s, l, b_2) \in TPC''$,
- the device constraints, i.e.
$$PRCR(OBA(ob)) = DVCR(dv)$$
for all objects $ob \in OB$ and devices $dv \in DV$ with $DVOB(dv) = ob$,
- the bead type constraints, i.e.
$$BA''(b) \in PR \Leftrightarrow BT''(b) = p \wedge$$
$$BA''(b) \in DV \Leftrightarrow BT''(b) = d \wedge$$
$$BA''(b) \in CM \Leftrightarrow BT''(b) = c$$
for all beads $b \in B''$,

- the bead replication constraints, i.e.

$$BA''(b_1) \neq BA''(b_2)$$

  for all $b_1, b_2 \in B''$ with $RPOB''(BOB''(b_1), BOB''(b_2))$,

- the mutual exclusion constraints for blocks, i.e.

$$BLE(bl_1) < BLS(bl_2) \vee BLE(bl_2) < BLS(bl_1)$$

  for all $bl_1, bl_2 \in blme$, where $blme \in \mathcal{P}(BL)$ is a given set of blocks that should be executed under mutual exclusion

- the no-overlap constraints, i.e.

$$BE''(b_1) < BS''(b_2) \vee BE''(b_2) < BS''(b_1)$$

  for all $b_1, b_2 \in B''$ with $BA''(b_1) = BA''(b_2)$, and

- the device type constraints, i.e.

$$NUDV(cr, dvt) \leq NADV(cr, dvt)$$

  for all crates $cr \in CR$ and device types $dvt \in DVTS$.

□

5 Conclusions

In this paper a brief survey is made of several hard real-time application domains, and important timing characteristics of the applications are inventarised (Section 2). It is discussed how these timing characteristics are translated into TPCs (Section 3).

Furthermore, a model is given, in which the hardware (Section 4.1), software (Section 4.2), and timing behaviour (Definition 4.15) of real-time systems can be specified. Based on this model, a general scheduling model for real-time systems is formulated (Section 4.3). In short, one has to find an assignment of a software specification onto a hardware architecture. The software description specifies a number of objects, which consist of non-preemptable beads (Definition 4.10). Beads together with TPCs constitute an execution graph (Definition 4.21). This graph should be extended such that communication is possible (Section 4.2.9). A schedule determines a location and a start time for each bead in an execution graph.

A schedule has to meet a number of constraints: the object bead constraints (Definition 4.12), the timed precedence constraints (Definition 4.15), the device constraints (Definition 4.18), the bead type constraints (Definition 4.20), the bead replication constraints (Definition 4.27), the no-overlap constraints (Definition 4.34), and the device type constraints (Definition 4.37).

6 Future research

The entire scheduling problem can be considered to consist of three main parts. First, the distribution of software across the hardware has to be determined, i.e. an assignment of objects to processors has to be given. Second, communication should be established. Third, start times have to be determined for all scheduling units. The entire problem is too complex to be solved in one step. Therefore, the problem has to be decomposed. To this end, there are several possibilities. It is the subject of further research to design effective decompositions and to solve the resulting subproblems. One solution strategy for the distribution and communication subproblems is treated in [5].

Apart from this, there are some other interesting open problems. One is the influence of data dependency on the scheduling problem. For example, an if-statement in a DEAL program is translated into a branch in the execution graph. Only one of the branches will correspond to the actual execution of the DEAL program at a given point in time. Since it is not known in advance which branch will be chosen, the scheduling algorithm has to allow both possibilities. One possibility to deal with this is to schedule both branches independently. As a result, the resources allocated to the 'unused' branch will remain idle for the duration of the execution of the other branch. A more efficient solution may be possible, if part of both branches could overlap during allocation and scheduling. For example, it may be possible to create a block that contains beads of both branches. If the block is assigned to one processor, the amount of idle time may be less than in the case of an assignment, in which the beads of both branches are executed on different processors. This is because one branch will be inactive in reality, which results in idle time on the corresponding processor.

A Appendix: list of symbols

symbol	meaning	definition
A	set of arcs in execution graph	4.21
A'	set of arcs in extended execution graph G'	4.25
A''	set of arcs in transformed execution graph G''	4.29
A_{CM}	set of arcs in execution graph induced by communication	4.21
A'_{CM}	set of arcs in extended execution graph G' induced by communication	4.25
A_{TPC}	set of arcs in execution graph induced by timed precedence constraints	4.21
A'_{TPC}	set of arcs in extended execution graph G' induced by timed precedence constraints	4.25
A''_{TPC}	set of arcs in transformed execution graph G'' induced by timed precedence constraints	4.29
B	set of beads	4.10
B'	set of replicated beads	4.24
BA	bead assignment	4.11
BCM	communicating beads	4.16
BDS	amount of communicated data	4.16
BE	bead end time	4.14
BL	set of block	4.31
BLB	beads of a block	4.31
BLE	bead end time	4.32
$BLRS$	resources of a block	4.31
BLS	block start time	4.32
$BLTR$	block time requirement	4.32
BOB	object of bead	4.10
BPR	bead processor requirement	4.14
BS	bead start time	4.14
BT	bead type	4.19
$BUSCAP$	bus capacity	4.3
$BUSCR$	crate location of bus	4.2
CM	set of communication media	4.6
$CMCON$	connected resources	4.6
$CMCAP$	communication medium capacity	4.6
CR	set of crates	4.1
$CRBUS$	bus in crate	4.1
$CRDV$	set of devices in crate	4.1
$CRMM$	set of memory modules in crate	4.1
$CRPR$	set of processors in crate	4.1
$DVCAP$	device capacity	4.3
$DVCR$	crate location of device	4.2
$DVOB$	object managing device	4.17

symbol	meaning	definition
DVT	device type	4.4
$DVTS$	set of device types	4.4
G	execution graph	4.21
G'	extended execution graph for replication	4.25
G''	transformed execution graph	4.29
L	label function	4.22
L'	extended label function	4.26
L''	transformed label function	4.30
LI	set of links	4.5
$LICAP$	link capacity	4.5
$LIPR$	processors connected to link	4.5
$MMCAP$	memory module capacity	4.3
$MMCR$	crate location of memory module	4.2
$NADV$	number of available devices of a type	4.35
NR	number of replicas	4.23
$NUDV$	number of used devices	4.36
OB	set of objects	4.8
OB'	set of replicated objects	4.24
OBA	object assignment	4.13
$PRCAP$	processor capacity	4.3
$PRCR$	crate location of processor	4.2
$RPOB$	replicated objects	4.24
RS	set of resources	4.7
TPC	set of timed precedence constraints	4.15
V	set of nodes in execution graph	4.21
V'	set of nodes in extended execution graph G'	4.25

References

[1] D. Alstein and P.D.V. van der Stok. Hard real-time reliable multicast in the DEDOS system. Lincoln, New Hampshire, September 1993. Third International Workshop on Responsive Systems, Lincoln, New Hampshire, September 1993.

[2] C. Bickford, M.S. Teo, G. Wallace, J.A. Stankovic, and K. Ramamritham. A robotic assembly application on the spring real-time system. Umass Computer Science Technical Report 96-06, University of Massachusetts, January 1996.

[3] G.D. Carlow. Architecture of the space shuttle primary avionics software system. *Communications of the ACM*, 27(9):926–936, September 1984.

[4] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: from simple message diffusion to byzantine agreement. Technical Report RJ5244, IBM Research Lab, San Jose, CA, December 1989. An earlier version was published as RJ4540.

[5] A.F.W. Gouder de Beauregard. Object assignment in real-time systems; a case study in DEDOS. Master's thesis, Eindhoven University of Technology, August 1996.

[6] G. Essink, E. Aarts, R. van Dongen, P. van Gerwen, J. Korst, and K. Vissers. Scheduling in programmable Video Signal Processors. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 284–287, 1991.

[7] D.K. Hammer. Process-oriented development of embedded systems; modeling behavior and dependability. To be published, 1997.

[8] D.K. Hammer, P. Lemmens, E. Luit, O.S. van Roosmalen, P. van der Stok, and J. Verhoosel. DEDOS: A distributed environment for object-oriented real-time systems. Dallas, Texas, USA, October 1994. 1st Workshop on Concurrent Object-Based Systems (COBS) at the 6th IEEE Symposium on Parallel and Distributed Processing (SPDP), Dallas, Texas, USA, October 1994.

[9] H. Kasahara and S. Narita. Parallel processing of robot-arm control computation on a multimicroprocessor system. *IEEE Journal of Robotics and Automation*, 1(2):104–113, June 1985.

[10] P.E.R Lippens, J.L. van Meerbergen, A. van der Werf, W.F.J. Verhaegh, B.T. Sweeney, J.O. Huisken, and O.P. McArdle. PHIDEO: A silicon compiler for high speed algorithms. In *Proceedings of the European Conference on Design Automation*, pages 436–441, 1991.

[11] J.D. Schoeffler. Distributed computer systems for industrial process control. *IEEE Computer*, 17(2):11–18, February 1984.

[12] A.D.H. Thomas, M.G. Rodd, J.D. Holt, and C.J. Neill. Real-time industrial visual inspection: A review. *Real-Time Imaging*, (1):139–158, 1995.

[13] G.J.W. van Dijk. *The Design of the EMPS Multiprocessor Executive for Distributed Computing.* PhD thesis, Eindhoven University of Technology, 1993.

[14] O.S. van Roosmalen. DEAL: an object-oriented language for distributed real-time systems. Dana Point, CA, USA, October 1994. Int. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), Dana Point, CA, USA, October 1994.

[15] W.F.J. Verhaegh. *Multidimensional periodic scheduling.* PhD thesis, Eindhoven University of Technology, December 1995.

[16] J.P.C. Verhoosel. *Pre-Run-Time Scheduling of Distributed Real-Time Systems.* PhD thesis, Eindhoven University of Technology, 1995.

[17] B.J.J. Verhulst. Specification of time constraints for SPRINT. Master's thesis, Eindhoven University of Technology, April 1993.

[18] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak, and C.B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(11):1240–1255, October 1978.

[19] H.H.H. Wevers. Reliable HRT intra-node multipoint RPC in DEDOS. Master's thesis, Eindhoven University of Technology, October 1995.

Computing Science Reports **Department of Mathematics and Computing Science**
Eindhoven University of Technology

In this series appeared:

96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.
96/02	P. de Bra and A. Aerts	Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12
96/03	W.M.P. van der Aalst	Parallel Computation of Reachable Dead States in a Free-choice Petri Net, p. 26.
96/04	S. Mauw	Example specifications in phi-SDL.
96/05	T. Basten and W.M.P. v.d. Aalst	A Process-Algebraic Approach to Life-Cycle Inheritance Inheritance = Encapsulation + Abstraction, p. 15.
96/06	W.M.P. van der Aalst and T. Basten	Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
96/07	M. Voorhoeve	Structural Petri Net Equivalence, p. 16.
96/08	A.T.M. Aerts, P.M.E. De Bra, J.T. de Munk	OODB Support for WWW Applications: Disclosing the internal structure of Hyperdocuments, p. 14.
96/09	F. Dignum, H. Weigand, E. Verharen	A Formal Specification of Deadlines using Dynamic Deontic Logic, p. 18.
96/10	R. Bloo, H. Geuvers	Explicit Substitution: on the Edge of Strong Normalisation, p. 13.
96/11	T. Laan	AUTOMATH and Pure Type Systems, p. 30.
96/12	F. Kamareddine and T. Laan	A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.
96/13	T. Borghuis	Priorean Tense Logics in Modal Pure Type Systems, p. 61
96/14	S.H.J. Bos and M.A. Reniers	The I^2C-bus in Discrete-Time Process Algebra, p. 25.
96/15	M.A. Reniers and J.J. Vereijken	Completeness in Discrete-Time Process Algebra, p. 139.
96/17	E. Boiten and P. Hoogendijk	Nested collections and polytypism, p. 11.
96/18	P.D.V. van der Stok	Real-Time Distributed Concurrency Control Algorithms with mixed time constraints, p. 71.
96/19	M.A. Reniers	Static Semantics of Message Sequence Charts, p. 71
96/20	L. Feijs	Algebraic Specification and Simulation of Lazy Functional Programs in a concurrent Environment, p. 27.
96/21	L. Bijlsma and R. Nederpelt	Predicate calculus: concepts and misconceptions, p. 26.
96/22	M.C.A. van de Graaf and G.J. Houben	Designing Effective Workflow Management Processes, p. 22.
96/23	W.M.P. van der Aalst	Structural Characterizations of sound workflow nets, p. 22.
96/24	M. Voorhoeve and W. van der Aalst	Conservative Adaption of Workflow, p.22
96/25	M. Vaccari and R.C. Backhouse	Deriving a systolic regular language recognizer, p. 28
97/01	B. Knaack and R. Gerth	A Discretisation Method for Asynchronous Timed Systems.
97/02	J. Hooman and O. v. Roosmalen	A Programming-Language Extension for Distributed Real-Time Systems, p. 50.
97/03	J. Blanco and A. v. Deursen	Basic Conditional Process Algebra, p. 20.
97/04	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra: Absolute Time, Relative Time and Parametric Time, p. 26.
97/05	J.C.M. Baeten and J.J. Vereijken	Discrete-Time Process Algebra with Empty Process, p. 51.

97/06	M. Franssen	Tools for the Construction of Correct Programs: an Overview, p. 33.
97/07	J.C.M. Baeten and J.A. Bergstra	Bounded Stacks, Bags and Queues, p. 15.
97/08	P. Hoogendijk and R.C. Backhouse	When do datatypes commute? p. 35.
97/09	Proceedings of the Second International Workshop on Communication Modeling, Veldhoven, The Netherlands, 9-10 June, 1997.	Communication Modeling- The Language/Action Perspective, p. 147.