ON THE MODELLING OF DISTRIBUTED REAL-TIME CONTROL APPLICATIONS

Martin Törngren

DAMEK-Mechatronics, Dept. of Machine Design,
The Royal Institute of Technology, S-100 44 Stockholm, Sweden.
Email: martin@damek.kth.se, Fax: +46-8-20 22 87

ABSTRACT: For the successful design and implementation of distributed real-time control applications, models that adequately state real-time behavioural requirements and provide information needed to assess different decentralization approaches are essential. Real-time behavioural models for control applications based on precisely time-triggered actions, synchronous execution and the specification of multirate interactions are introduced. Contemporary computer science models are subsequently evaluated. The use of the modelling approach in design of distributed control systems is discussed.

Keywords: Distributed control, Control applications, Distributed computer control systems, Real-time systems, Modelling.

1. INTRODUCTION

Technological progress and user needs are stimulating new areas of distributed computer control systems (DCCS). In a historical perspective, DCCS were first introduced in process control applications in the 70s, followed by manufacturing systems, and in the late 80s/ early 90s by *machine embedded control systems*. Further, related to this there is the mainstream of general purpose distributed processing and telecommunication systems. In view of the multitude of available distributed system concepts and applications it is clearly important and natural to focus on the characteristics and requirements of a particular applications class when considering a DCCS approach. In this context and in particular for distributed systems, modelling is very important.

This paper is motivated by distributed real-time control applications as found or being developed in for example vehicles, aircrafts, robots and process control. Even though such distributed applications have things in common with previous distributed applications, real-time requirements are different. Moreover, the inherent interdisciplinarity of real-time control applications constitutes an important problem for DCCS design.

Modelling is necessary to capture application requirements and to support the design process from specification to implementation. I.e. the modelling approach should support successive refinements and be formal enough to provide a basis for automated tool support. For real-time control applications it is important that models support *inter-disciplinary design* (e.g. necessary cooperation between computer and control engineers). Key issues with respect to decentralization need modelling support. A DCCS implementation involves fundamental issues concerned with how to find a suitable *mapping* between an application and the underlying architecture. There are then many degrees of freedom for a designer with respect to implementation oriented application structuring, allocation and the choice of execution strategy. Such choices can only be found and evaluated if the application is modelled appropriately.

Paper purpose and structure: The paper focuses on real-time behavioural models of control applications. Section 2 reviews essential application characteristics. Section 3 presents a brief survey of existing real-time modelling approaches. Section 4 presents modelling extensions for real-time control applications and indicates their use in design. Section 5 gives conclusions.

2. CHARACTERISTICS OF REAL-TIME CONTROL APPLICATIONS

Control applications are often described using modes and a hierarchical structure. For each mode "top-level" functions that provide system services can be identified, compare for instance with the functions *accelerate/decelerate* in an automobile. These involve an interaction between several subsystems (decomposed functions), e.g. engine, braking, transmission and clutch control. A top-level function may therefore not have a clear correspondence to a particular unit or subsystem. Top-level functions may have different characteristics depending on the particular mode. This could relate to for example sampling frequencies or control algorithms used. Top-level functions can be further decomposed in a hierarchical fashion until *elementary functions* have been obtained.

2.1 Control, data-flow and multirate interactions

There are many names used to denote a thread of control in a computer system, including usecase, process, task, thread of control, transaction, etc. In this paper the term *activity* will be used to denote control-flow which may stretch over several elementary functions. It is important to distinguish between a specified activity and its implementation. In the implementation, an activity may involve the execution of sequences of elementary functions in several nodes. These entities are referred to as processes.

A simple control system is characterized by few couplings between elementary functions. In more complex systems, elementary functions can be involved in the provision of several top-level functions, and often performed with different sampling rates, referred to as multirate systems. Data-flows between elementary functions typically have a connectivity of one-to-one or one-to-many. Data-flows take place within a thread of control and between threads of control with equal or different periods. Such interactions between threads of control with different periods are referred to as *multirate interactions*, Törngren (1995). A specific problem concerns how to model and treat such interactions. Consider for example a multiple drive system in a paper machine where there is a need to maintain a strict tension of the paper, thus requiring associated drives to be sufficiently synchronized. There is a local controller per drive and a coordinating controller that based on feedback from the local controllers maintains the synchronization by providing speed references to the local controllers. If the system is multirate it is typically desirable to specify the control delays (constant or bounded) of the data-flows between local and coordinating controllers, in order to fulfil control objectives. A control delay can be principally defined as the delay from sampling to actuation.

Data-flows include discrete (e.g. system mode) and continuous (e.g. shaft position) valued data. For discrete data (e.g. limit switch on or off, system mode one to five) a value change is very significant. Continuous data (e.g. shaft position or position reference) is a representation of a continuous entity which is discretized for use in the computer, e.g. by analog to digital conversion or by computation. The difference between two consecutive samples in this case depends on the bit resolution (giving a quantization error), the sampling frequency and the measurement accuracy (in case of sensor sampling). With sufficient resolution, sampling frequency and low noise, the difference is usually much less significant compared to the former discrete case. These two types of data-flows are referred to as discrete vs. continuous.

2.2 Timing requirements

Control applications often involve both sampled data control and sequence control (event-response control). The control system thus needs to handle external events (periodically or aperiodically) and perform periodic control. The triggering of activities/elementary functions is expressed using different terminology in various disciplines. In computer science and in control theory the following terms are used respectively: periodic (time-triggered)/aperiodic (event-triggered), Kopetz et al. (1991) vs. continuous/discrete/discrete event systems. These terms are similar but there are some important discrepancies. The most common interpretation of periodic execution in computer science is an execution that takes place some time during its period. This has the effect to introduce period variations (termed jitter). Periodic operation as understood in control theory, however, is interpreted to mean periodic sampling and actuation with zero jitter, perfect synchronization in multirate systems and constant control delays (also for multirate interactions), Wittenmark et al. (1995), Törngren (1995). There is thus a gap between real-time control applications and computer science models.

From discrete time control theory the following three requirements are readily identified: *Periodic actions with equidistant intervals, Synchronized actions,* and *Response time,* see Åström and Wittenmark (1990). Whereas, disturbances, model inaccuracies and sensor noise, have been extensively treated at the controlled process side, little work has treated deficiencies in the computer system implementation of the control system with respect to *time-variations*, Wittenmark et al. (1995), one exception being response times. Time-variations can refer to varying *sampling periods,* i.e. varying intervals between successive sampling actions, *varying control delays,* i.e. varying intervals between related sampling and control actions, or *non* (perfectly) *synchronized sampling and actuation* in multivariable and multirate control systems. Variations are among other things due to run-time scheduling and varying execution times. Time-variations have the effect to deteriorate control performance and can potentially cause instability, Törngren (1995).

To capture behavioural requirements of sampled data control systems, the concept of *"Precisely time-triggered actions"* with *tolerances* was introduced by Törngren (1995). They are used for specifying the timing behaviour of sampling and actuation actions (corresponding to a realistic interpretation of the assumption that such actions occur exactly at specified time instants). The applicability of precisely time-triggered actions with tolerances is however believed to be larger and can be introduced for both relative and absolute timing constraints. Kopetz and Kim (1990) used a similar notion to manage consistency constraints in distributed systems (update of discrete data, simultaneous mode changes). For sampled data applications a relative accuracy (bounded clock drift) is sufficient. The tolerances can also be interpreted as synchronization requirements (precision) for precisely time-triggered actions in multivariable control systems, e.g. synchronizing actuation of several outputs. *Tolerance specifications* can be used to define the allowed deviation from nominal timing. For automatic control applications, tolerances can in principle be derived from control analysis, Törngren (1995).

3. APPLICABILITY OF PROPOSED MODELS FOR REAL-TIME CONTROL SYSTEMS

Other surveys of modelling approaches can be found in Lawson (1991), Motus and Rodd (1994) and Bucci et al. (1995). Modelling approaches are here categorized based on their origin, or essential aspect.

Structured methods based on control and/or data flow. Examples in this category include the models used in the MARS system, Kopetz et al. (1991), HRT-HOOD, Burns and Wellings (1994) and GRAPE, Lauwereins et al. (1995), and DARTS/DA, Gooma (1989). This category can be further subdivided into control and data flow oriented models. Models with a basis in real-time scheduling theory are characterized by a focus on control flow and explicit considerations of timing specifications, although with an emphasis on response time requirements. Methods in structured analysis have been extended towards real-time systems by introducing also control-flow aspects in data-flow graphs, e.g. Hatley and Pirbhai (1987). The methods only in a limited fashion consider timing specifications and further mix control- and data-flow specifications which reduces clarity.

Finite automata and extensions. Examples in this category include Modecharts, Jahanian et al. (1988) and Petri-Nets and their extensions, Bucci et al. (1995). These models focus on system behaviour. However, real-time is most often treated as an extension. The Real Time Logic used in Modecharts is an exception in this regard and allows the specification of both relative and absolute timing of events. However, only response time requirements are considered in given examples. Petri-nets are powerful, but have been criticized for not scaling up to larger applications and for making modelling complex, Motus and Rodd (1994).

Formal methods. Examples include CSP, Hoare (1978), the Q-model, Motus and Rodd (1994), and synchronous programming languages, Halbwachs (1993). One major problem with the majority of these approaches is their focus on non real-time systems and the problems of later incorporating real-time. The synchronous programming languages provide a family of models, with a state or data-flow style. In the models, actions are considered to take "no observable time", e.g. instantaneous broadcasts of data are considered. This facilitates verification of qualitative temporal behaviour. Specification of timing requirements is possible but the specification of multirate interactions is unclear. The Q model is discussed below.

Models in automatic control and signal processing. The models in this category describe functional (transformations), behavioural aspects and data-flow in terms of discrete-time equations. Timing specifications and assumptions were briefly discussed in Section 2.

Bucci et al. (1995) classifies real-time models into *operational* (essentially state machine based, compare with the two first groups) and *descriptive* (based on mathematical notations, compare with the two last groups) approaches. This corresponds well to the above classification. The surveyed groups are typically good at some, but not all of the *structural, functional and behavioural* aspects, that are all essential for modelling of real-time systems. A few selected modelling approaches are now further discussed.

3.1 Precisely time-triggered actions

The fact that other timing requirements than response times have largely been neglected, is being realized to some extent in the "real-time computer science community", see e.g. Locke (1992), Lawson (1991), Motus and Rodd (1994). A pragmatic approach to implement precise time-triggering (and to reduce time-variations) in priority based scheduling is to create a separate "high" priority process that performs jitter sensitive actions. Since the priority of the process is relatively high, the time-variation can be made small.

In static scheduling, release time and deadline attributes for tasks, $\{R, D\}$, are used, Xu and Parnas (1990). A precisely time-triggered action can be translated into these attributes according to a given tolerance specification. If precedence relations are added, i.e. as control flow specifications over a number of elementary functions (objects, processes, etc.), a sequence of time-triggered actions can be described. To describe a constant delay, consider a control system performing sampling (C_s), computation (C_c) and actuation (C_a), with execution times given in parenthesis. Assume that the same tolerance, *Tol,* is given on the period, *T,* and the control delay, t_c. This activity then needs to be modelled as the following three precedence related processes:

Sampling process: $\{C_s, R_s=0, D_s<Tol, T\}$
Computation process: $\{C_c, D_c<t_c, R_c=Tol, T\}$
Actuation process: $\{C_a, R_a=t_c, D_a<t_c+Tol, T\}$.

Release times and deadlines are in the example used to enforce the precedence relations as well as the two precisely time triggered actions. The combination of precedence relations and $\{R, D\}$ attributes, for each process, only seems to have been applied in static scheduling. Limited attempts in this direction for priority based scheduling are described by Audsley et al. (1995). However, deadlines (shorter than periods) and release times have been used in process models for priority based scheduling, Audsley et al. (1995). Similar to the example, this model can be used to *implement* precedence relations and precisely time-triggered actions. There are also response time oriented approaches based on precedence graphs, used together with both static and priority based scheduling. A deadline is in this case associated with the complete precedence graph, compare e.g. with HRT-HOOD, Burns et al. (1994). The MARS system scheduler, Fohler (1994) guarantees that a period specification corresponds to the implemented period.

While release times, deadlines and precedence relations are useful for implementation, they are definitely not ideal for specification of control systems. Further, *time synchronized* actions are not considered by these approaches. Synchronization, if provided in computer science models, most often refer to resource synchronization, blocking communication or qualitative timing properties. One notable exception with regard to precisely time-triggered and synchronized actions is the Q-model, Motus and Rodd (1994). A group of synchronous and time-triggered functions is referred to as a synchronous cluster. A time trigger is defined in terms of a "null channel" and the activation is specified as follows: The *equivalence interval* defines the maximum response time for functions within a cluster, counted from the time-trigger. The *simultaneity interval* defines the maximum time difference in the activation of functions within a cluster. *Tolerance interval* is basically used as in this paper. The simultaneity interval is supposed to be derived from the tolerance by subtracting a safety margin.

3.2 Evaluation with respect to multirate interactions

Multirate interactions hardly seem to have been addressed at all in computer science models. Delays in communication between pairs of periodic functions have been investigated by Motus and Rodd (1994), but not in the context of multirate sampled data systems. One recent attempt in this direction is the introduction of so called end to end deadlines/delays, Klein et al. (1994), Tindell and Clark (1994). Although not explicitly considered for multirate interactions, an end to end deadline can be associated with the data-flow between activities with different rates. However, unless time synchronized execution between activities is considered, the approach can only be used to specify bounded delays, further discussed in section 4.

A rather similar approach could be based on a specification of the allowed "age" of data in multirate interactions. Although the notion of data age has been used (primarily in relation to consistency) it does not appear to have been used in relation to multirate interactions. Another approach is to consider a time-triggered precedence graph which has a period equal to the largest common divisor (*LCD*) of periods. The elementary functions part of the graph are specified to execute at an integer multiple of the *LCD*. While this model is simple it is also limited since restrictions on the execution times of functions are introduced (each function must complete within the *LCD*). A similar technique is well known in signal processing, allowing operations like down- and up-sampling. In the GRAPE system, for digital signal processing applications, Lauwereins et al. (1995), multirate relations of this type are considered, referred to as "merged synchronous execution". GRAPE is based on extended data-flow models in which timing requirements are oriented towards bounded delays.

4. MODELLING EXTENSIONS FOR REAL-TIME CONTROL APPLICATIONS

Based on the above discussion it can be concluded that most current real-time computer science oriented modelling approaches do not adequately address issues pertinent to control applications. Limitations exist primarily with respect to behavioural aspects. Modelling extensions are sketched in the following.

4.1 Requirements on a model supporting design of distributed control applications.

An important aim with the model is to be able to predict the consequences of applying a particular decomposition, partitioning and allocation strategy with respect to resource utilization and timing. The model therefore needs the following information:

Data-flow (structural view): The connections between elementary functions and data sizes must be specified.

Execution requirements (functional view): For all relevant elementary functions the resource requirements must be stated, or possible to derive, i.e. execution time (or e.g. the number and type of arithmetic operations required by algorithms) and memory requirements.

Control flow and timing requirements (behavioural view): The control-flow specifies when a function executes. Relevant timing on data- and control-flow must be specified, including periods, control delays, and tolerances for (synchronized) precisely time-triggered actions.Further, most typically a designer would like to specify optimization directives for sampling periods and control delays, e.g. to provide a minimized control delay.

Dependability. Dependability requirements are important but outside the scope of the paper.

With the above specifications it is possible to derive estimates of real-time processing and communication requirements (e.g. bytes/s and operations/s). With an architectural model and relevant parameters, these estimates can be translated into execution and communication times. Load balancing is easily assessed. It is also possible to estimate delays of data- and/or control-flow as a function of communication, execution and expected overheads. It should be noted that the derivation of execution time estimates is a problem for dimensioning, in particular early in design.

4.2 Real-time behavioural model extensions

To capture essential behavioural requirements of control applications, two modelling ingredients are introduced, namely *elementary function triggers* and *multirate interaction* descriptions. The elementary function is the basic entity used in the -following discussion. It constitutes the basic entity used in modelling and design.

Triggers for elementary functions. Starting triggers are defined for elementary functions A trigger can be a time-trigger (referring to periodic triggering) or an event-trigger. Time-triggers can further be defined with a tolerance as introduced in section 2.2. Functions execute when triggered and can be associated with a computational model: *Read input channels; Perform computations; Write the computed output to the output channels.* It is thus assumed that the *read* action is performed at the beginning of the function, and that the function ends by *writing* its result. A single elementary function with a time trigger and a tolerance can be used to specify periodic sampling. A sequence of two elementary and precisely time-triggered functions can be used to specify a constant delay from sampling to actuation for a simple sampled data system. The second trigger is in this case defined with a constant delay relation to the first trigger and the tolerance for the second trigger refers to this constant delay.

Synchronous execution. Two or more periodic elementary functions (activities) execute *synchronously* when related period start points of the functions always are separated by a known constant called *phase*, within a specified tolerance. Synchronization can be provided by a global clock or by explicit synchronization. For synchronous execution in multirate systems the phase refers to the least common multiple of periods - thus interpreting "related period start points". Consequently, in *asynchronous* execution the period start points of activities do not have a guaranteed relation, typically varying depending on the characteristics of the local clocks.

Explicitly separated data- and control-flow specifications. An activity specifies the control-flow over several elementary functions by use of a precedence graph, function triggers and, if required, explicit deadline requirements. A precedence relation between elementary functions A and B corresponds to B being event-triggered by the completion of A. For control applications it is common that control and data flow coincide. However, the separation of control and data-flow specifications is still preferred for clarity and to be able to appropriately model multirate interactions.

Multirate interactions. For multirate systems the delays of data-flows between activities with different rates need to be specified. Three modelling approaches, *overlapping, extended separate* and *merged* are introduced for this purpose by Törngren (1995). The approaches are best illustrated using an example: a small multirate control system, see Fig. 1.

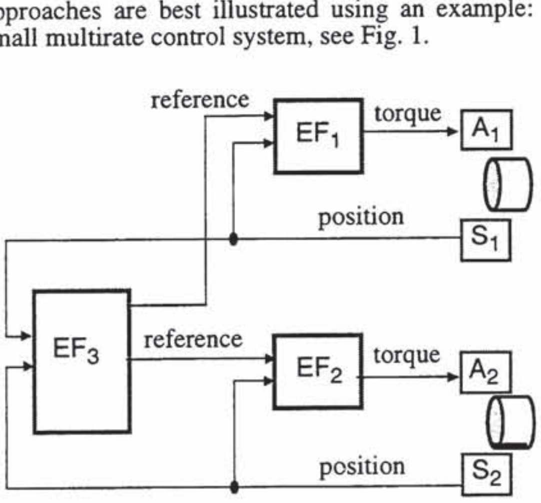

Fig. 1. Data-flow graph of the example system.

The system involves two "local" elementary functions, EF_1 and EF_2, corresponding to two motor servos, and one coordinating function, EF_3 that performs feedback based synchronization. Correspondingly, three activities can be identified. It is assumed that EF_1 and EF_2 have period T and EF_3 period T_3. The interfacing functions performing sampling and actuation are denoted S_i and A_i (for $i=1,2$). $S_{i:1..2}$ (similarly for $A_{i:1..2}$ and $EF_{i:1..2}$) is used to denote S_1 and S_2

Control-flow and timing specifications using the extended separate and overlapping approaches are now illustrated for the coordinating controller. The graphical notation used is shown in Fig. 2, which is a specification of the local controllers. Circles are used to denote elementary functions (control algorithms, samplers and actuators). Fig. 2 illustrates the use of time-triggers (TT), trigger source (*CLK*), period, and for the second trigger also a delay (t) with a minimization directive (<) and a tolerance specification (*tol*). Note that the same delay is specified for both local activities and that the local controllers are specified to execute synchronously. Alternatively, if different delays are required, the controllers are specified separately. If the local controllers had been synthesized using continuous time design, the time-trigger for actuation could be removed and possibly replaced by an explicit deadline specification (bounding the control delay sufficiently below T). Horizontal arrows are used to denote precedence relations and vertical arrows time-triggers. In case a function has both a precedence relation and a time-trigger, the latter specifies when execution occurs and the former is interpreted as a specification: The preceding functions and data-flows should be performed prior to the time-trigger.

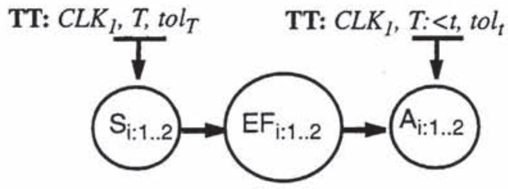

Fig. 2. Control-flow spec. of local functions.

The basic idea in the *extended separate approach* is to model activities separately and to introduce *rate interfacing functions (RIFs)* which ascertain that the delays between activities are bounded. Computer science models that have considered the possibility of multirate interactions tend to fall in this category. Extended separate modelling is used in Fig. 3 for the coordinating function.

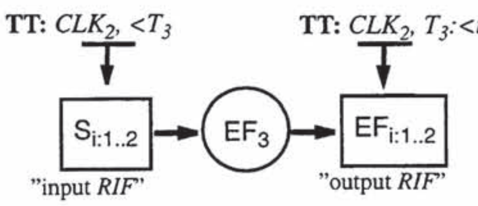

Fig. 3. Extended separate modelling of coord.function.

Boxes (in control-flow graphs) are used to denote *RIFs*. These functions retrieve or provide data from or to an elementary function that belongs to another activity (and which has another period). *RIFs* are used together with timing specifications for an activity to ensure that that control delays are bounded. The input *RIF* is interpreted as the retrieval of data produced by functions S_1 and S_2 ($S_{i:1..2}$). Since the activities in this approach are asynchronous (i.e. coordinating vs. local) each data item that is retrieved may have been created up to one sampling period, T (period of data source) ago. The output RIF is interpreted as the provision of data to functions EF_1 and EF_2 ($EF_{i:1..2}$). There is a detection delay of up to T, associated with the delivery. For the coordinating activity, the control flow specification can thus be interpreted as follows: Retrieve data from $S_{i:1..2}$, execute EF_3 and provide data to $EF_{i:1..2}$ according to the given timing specifications. Due to asynchronous execution, a time-varying but bounded delay (<$2T$ in this example) is introduced. Consequently, the extended separate style of modelling is sufficient only when a time-varying delay is acceptable.

When constant delays in multirate interactions are required (complying with sampled data models), the *overlapping approach* is suitable provided that period ratios are integers. A hierarchical view in terms of a sampling frequency hierarchy of the system is utilized.

The idea is to specify an activity at a particular level by including all elementary functions at the same and lower layers. As a consequence, an elementary function can be used in several activity specifications (hence overlapping). A function can because of this be associated with several sets of timing requirements which must be resolved during scheduling. Overlapping modelling is exemplified in Fig. 4.

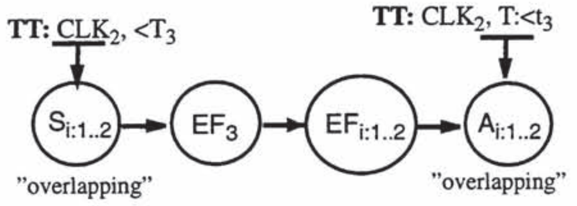

Fig. 4. Overlapping modelling of coord. function.

When period ratios are rational numbers, *merged modelling* may need to be applied. In this approach, all related (communicating) activities are modelled together in one precedence graph as one merged activity. To do this, the precedence graph must be extended to the least common multiple of the periods with which the merged activity is periodic.

4.3 Design of distributed applications using the model

The introduced modelling concepts, *function triggers* (precisely time-triggered actions), *synchronous execution* and approaches for specifying *multirate interactions*, can be used together with the other more traditional concepts like precedence graphs, data and control-flow specifications to describe the real-time behaviour of control application. The model emphasises the fact that a control application can be described in terms of two top-level timing requirements: *Precisely time-triggered actions,* and *End to end deadlines.* The latter arises for constant delay specifications, in an activity that includes more than one precisely time-triggered action. Once the instants of precisely time-triggered actions have been fixed, phase relations can be interpreted as end to end deadlines. End to end deadlines also appear for conventional response type activities.

During design, the system model can be successively refined while the overall timing requirements are maintained. For example, during system decomposition more functions and data-flow paths are added. During evaluation of different allocation approaches, assessment can be done with regards to resulting communication and execution load.

The model is believed to be essential for execution strategy considerations, used to refer to a set of selected policies for triggering, synchronization and scheduling. These policies have a major impact on the timing of a system. When comparing a desired timing behaviour with constraints imposed by design situations, the system hardware architecture and resource management policies may be more or less fixed, thus constraining the allowable solution space. Therefore, in particular when subsystems are provided by different vendors, it is important to define suitable system design policies and interfaces, which typically are at higher layers than normally considered by communication standards and protocols. As an example, consider the implementation of the coordinating function in a distributed system. This means that the precedence graph of Fig. 4 will be split in e.g. two parts yielding two processes. To achieve the constant delay specified in Fig. 4, synchronous execution is required. If this is not possible, various transformations to reduce the induced time-variations are possible, e.g. to change the latter trigger from time-triggering to an aperiodic server of some type. As discussed in section 3, the model can be used to assess the applicability of scheduling approaches. These issues are further discussed by Törngren (1995).

5.CONCLUSIONS

The paper has described essential modelling ingredients for distributed real-time control applications. The modelling concept has been applied in an industrial robot case study, Törngren (1995), and so far promises to be a useful approach.

6.REFERENCES

Audsley N.C., Burns A., Davis R.I., Tindell K.W., Wellings A.J. (1995). Fixed priority pre-emptive scheduling. *J. Real-Time Systems*, 8, 173-198. Kluwer.
Bucci G., Campani M., Nesi P. (1995). Tools for specifying real-time systems. *J. Real-Time Systems*, 8, 117-172.
Burns A., Wellings A.J., (1994). HRT-HOOD: A Structured Design Method for Hard Real-Time Systems. *J. of Real-Time Systems*, Vol. 6, No. 1, January 1994.
Fohler G. (1994). *Flexibility in statically scheduled hard real-time systems.* PhD thesis, Technische Universität Wien, Institut fur Technische Informatik. April 1994.
Gomaa H. (1989). A software design method for distributed real-time applications. *J. of Systems and Software,* 9, 81-94. Elsevier Science Publishing.
Halbwachs N. (1993). *Synchronous programming of reactive systems.* ISBN 0-7923-9311-2. Kluwer.
Hatley D.J., Pirbhai I.A. (1987). *Strategies for real-time system specification.* Dorset House Publ., New York.
Hoare C.A.R. (1978). Communicating Sequential Processes. *Comm. ACM,* Vol. 21, No. 8, pp. 666-677.
Jahanian F., Lee R., Mok A. (1988). Semantics of Modechart in Real Time Logic. *Proc. of 21st Hawaii Int. Conf. on Systems Sciences*, pp 479-489, Jan. 1988.
Klein M.H., Lehoczky J.P., Rajkumar R (1994). Rate-Monotonic Analysis for Real-Time Industrial Computing. *IEEE Computer,* January, pp. 24-32.
Kopetz H., Zainlinger R., Fohler G., Kantz H., Puschner P., Schutz W. (1991). The design of real-time systems: From specification to implementation and verification. *IEE Software Eng. J.,* 6(3), pp. 72-82, May 1991.
Kopetz H., Kim K. (1990). Real-time temporal uncertainties in interactions among real-time objects. *Proc. 9th IEEE Symposium on Reliable and Distributed Systems,* Huntsville, AL.
Lauwereins R., Engels M., Ade M., Peperstraete J.A. (1995). Grape-II: A system-level prototyping environment for DSP applications. *IEEE Computer,* Feb. 1995, pp. 35-43.
Lawson H. (1991). *Parallel Processing in Industrial Real-Time Applications*, Prentice Hall.
Motus and Rodd (1994). *Timing analysis of real-time software.* Pergamon (Elsevier science).
Tindell K. and Clark J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming,* 40:117-134.
Törngren M. (1995). *Modelling and Design of Distributed Real-Time Control System.* PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Sweden.
Wittenmark, B., Nilsson J., Törngren, M. (1995). Timing Problems in Real-time Control Systems: Problem Formulation. *In Proc. of the American Control Conf.*
Xu J., Parnas L.P. (1990). Scheduling processes with release times, deadlines, precedence and exclusion relations. *IEEE Trans. on Software Engineering,* 16, 360-369.
Åström K. J., Wittenmark B. (1990). Computer controlled systems, theory and design, 2:nd edition, 1990, Prentice Hall. ISBN 0-13-172784-2.