ELSEVIER

Journal of Systems Architecture 45 (1998) 131–156

JOURNAL OF SYSTEMS ARCHITECTURE

Network conscious design of distributed real-time systems [1]

Jung Woo Park [a], Young Shin Kim [a], Seongsoo Hong [a], Manas Saksena [b], Sam H. Noh [c], Wook Hyun Kwon [a,*]

[a] *School of Electrical Engineering and ERC-ACI, Bldg. 133 Autom. Syst. Res. Inst., Seoul National University, Seoul 151-742, South Korea*

[b] *Department of Computer Science, Concordia University, Montreal, PQ H3G IM8, Canada*

[c] *Department of Computer Engineering, Hong-Ik University, Seoul 121-791, South Korea*

Received 20 December 1996; received in revised form 10 July 1997; accepted 5 September 1997

Abstract

In this paper, we present a network conscious approach to designing distributed real-time systems. Given a task graph design of the system, the end-to-end constraints on the inputs and outputs, and the task allocation on a given distributed platform, we automatically generate task attributes (e.g., periods and deadlines) such that (i) the task set is schedulable, and (ii) the end-to-end constraints are satisfied. The method works by first transforming the end-to-end constraints into a set of intermediate constraints on task attributes, and then solving the intermediate constraints. The complexity of constraint solving is tackled by reducing the problem into more tractable parts, and then solving each subproblem using heuristics to enhance schedulability. The methodology presented in this process can be mostly automated, and provides useful feedback to a designer when it fails to find a solution. We expect that the techniques presented in this paper will help reduce the laborious process of designing a real-time system, by bringing resource contention and schedulability aspects early into the design process. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Distributed real-time systems; Design methodology; Real-time network; Constraint solving; Nonlinear optimization; Real-time scheduling; Control system

1. Introduction

Distributed computing is an eminently important requirement in a large scale industrial computer system where hundreds of heterogeneous

[*] Corresponding author. E-mail: sshong@redwood.snu.ac.kr

[1] The work reported in this paper was supported in part by Engineering Research Center for Advanced Control and Instrumentation (ERC-ACI) under Grant 96K3-0707-02-06-1, by KOSEF under Grant 96-2037, and by NSERC Operating Grant OGP0170345.

control computers such as digital signal processors and process logic controllers are operating with thousands of custom input and output devices. As demands for factory automation increase, industrial distributed computer systems are required to integrate even more diverse computing elements such as supervisory workstations, database servers, and client stations. As a result, a distributed system in a factory now needs to provide interoperability among a wide variety of computers from control processors spread on the factory floor to workstations that operate in factory offices.

Often, such heterogeneous and complex industrial distributed systems are configured into a layered architecture, the layers being called *application* and *field layers*. At the application layer, engineering workstations are connected through standard communication networks such as the Ethernet. The computers at this layer are used for supervising control processes and executing management applications, and offer a distributed supervisory and information processing infrastructure. On the other hand, the field layer connects special purpose control processors which communicate with input and output devices via industrial real-time networks. Computations at this layer are subject to a variety of correctness and performance criteria such as timeliness, responsiveness, and reliability. This is because field layer applications are time-critical in the sense that each application has stringent timing constraints, and failure to meet such constraints may yield catastrophic results. Examples of such constraints are:

1. The pressure of a chemical reaction tank should be monitored and regulated every 10 ms.
2. Sensor readings of the system must propagate to the actuator within the sampling period of 35 ms.

Note that violating the first constraint may possibly cause a tank explosion. A software system which is subject to timing constraints is called a *real-time system.*

While it is possible to take advantage of recent advances of distributed systems for the design of the application layer, it is extremely difficult to design the field layer such that both performance requirements and software engineering requirements such as design traceability and scalability are guaranteed. The reason is that a time-critical distributed real-time system cannot be designed in a composable fashion since temporal relationships introduce coupling between structurally irrelevant components. For example, unpredicted overload of a component may prevent another from meeting its timing constraints regardless of their functional relationship. For the lack of design theory and associated tools, it is thus common practice to realize a field layer subsystem with ad hoc engineering approaches which rely on manual decomposition, tuning, and laborious validation. However, maintaining design composability is essential even in the presence of timing constraints.

A plausible solution to this problem is to map system-level timing constraints onto component-level timing constraints based on the functional structure of the system. With this approach, the development of a distributed real-time system involves simply implementing the constituent components, and then integrating them together. Of course, the component-level timing constraints collectively must guarantee the system-level timing constraints. This is an attractive methodology as recent advances in real-time scheduling and analysis techniques have established a sound theoretical basis for real-time systems, especially for those running in isolation or in a controlled environment. Thus, while timing constraints at the system-level may be difficult to satisfy, satisfying timing constraints at the component-level may not be a formidable task. Many industrial systems, especially those in the domain of distributed real-time systems have been modeled and analyzed using such scheduling theories. However, these techniques have been applied to industrial systems only in crude form [1,2].

Thus far, the component-level timing constraints (i.e., task periods, deadlines, etc.) that have often been handled by most techniques [3,4] are often artifacts of the way the system is designed where for a given system there may be many different ways to meet the end-user requirements. The choice made is often mandated by the programmer, and the resource constraints are rarely taken into account. Unfortunately, many of today's real-time systems require sharing of resources and data, synchronizing task executions, and timely flow of data through multiple data paths on a distributed platform. As systems become more complex, ad hoc methods to derive feasible task parameters lead to wastage of valuable development time and computer resources. There is also a serious problem in this practice: many of the key synchronization and precedence requirements become tightly coupled with the derived timing constraints. As a result, real-time programmers often lose the traceability of the system under development or maintenance, and significant redesign may be required if the timing constraints are changed.

The approach we take in this paper is, likewise, to obtain a runnable system design that guarantees the system-level constraints by satisfying the component-level constraints. However, unlike previous approaches, we take a systematic approach to the problem such that changes in timing or resource constraints do not require a major redesign of the system. Furthermore, resource

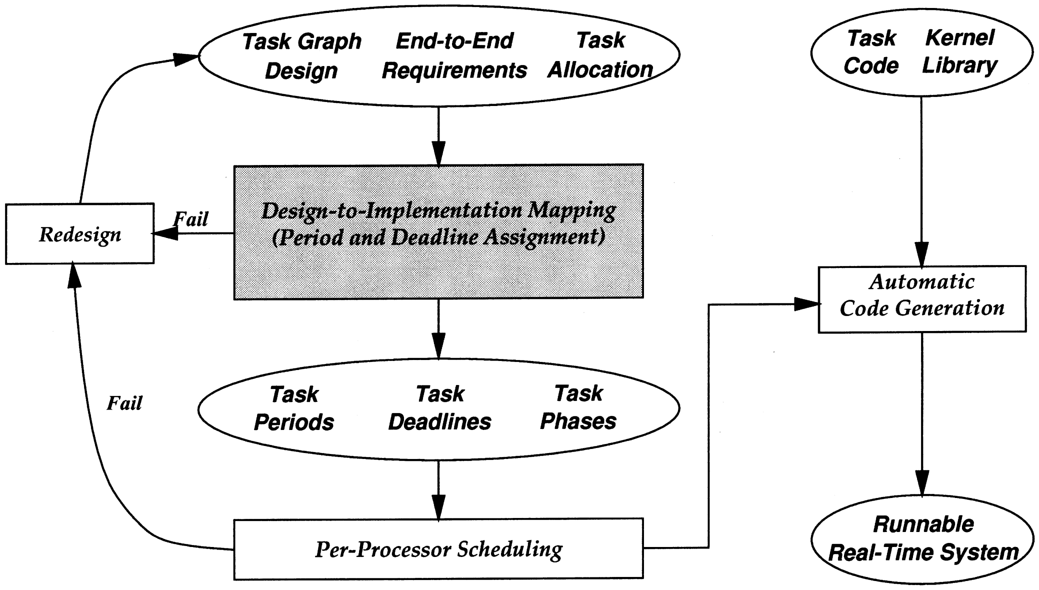


Fig. 1. The structure of the end-to-end design methodology.

constraints are taken into account from the start minimizing resource waste.

Fig. 1 shows the overview of our end-to-end system design methodology to generate a runnable system from a task graph design of the system. This process is composed of three phases. First, given a task graph design, the end-to-end constraints on the inputs and outputs of the system, and the task allocation on a given distributed platform, the task specific attributes are derived. Second, the derived task attributes are used to schedule the task, and finally, the user-provided task code is merged with the real-time kernel libraries to generate the runnable real-time system.

The focus of this paper is on the first phase of this process, that is, the *derivation of task specific attributes.* (This is specified by the shaded box in Fig. 1.) This problem is modeled as a constraint solving problem, in which the original end-to-end timing constraints are expressed as a set of intermediate constraints on task-specific attributes. The intermediate constraints are then solved to derive a set of schedulable task parameters. The second and third phases are not addressed here since real-time scheduling and automatic code generation techniques have been well addressed in the literature [5,6].

The main contribution of our work is in the development of a systematic methodology to transform a high-level design into a system that is schedulable. The methodology presented in this paper provides substantial benefits: (1) It provides designers with a rapid prototyping tool which will help in building a running prototype quickly. This, in turn, will help locate and isolate schedulability and performance bottlenecks. (2) It helps the designers fix and optimize a faulty design for both correctness and performance. This is possible not only because system traceability is maintained throughout the approach, but also because the constraint solver itself generates various performance metrics.

1.1. Related work

Two areas in real-time systems come close to our approach: (1) real-time software design methods and (2) real-time scheduling. There are a number of software design methodologies specifically aimed for complex, distributed real-time programming. In [7], Gomaa presents a method called DARTS (design method for real-time systems) which models a real-time system as a set of concurrent, communicating tasks. DARTS shares the same task model as our task graph, but it provides a more diverse set of communication and synchronization primitives such as message queues, mail boxes, and semaphores. Unlike our approach, DARTS does not take into account system scheduling, and it cannot provide any support for bounding blocking times of communicating tasks. It is thus the programmers' responsibility to guarantee bounded blocking times. Recently, Selic [8] also presented a design method called ROOM (real-time object oriented method) which supports high-level modular decomposition and interconnection between modules. Although these methods do not address the problem of managing end-to-end timing requirements of a real-time system, they can be used to generate a task graph design used in our approach.

Real-time scheduling has been a fertile area of research in the last decade. We refer the readers to [5,6] for an overview of real-time scheduling. However, there has been relatively little effort in the integration of design and scheduling, and specifically, the derivation of task periods and deadlines from end-to-end constraints. In [9], similar problems are formulated as a real-time database consistency problem, but the focus is more on schedulability analysis and less on the derivation of task parameters.

In [10], Gerber et al. first address the problem of systematic translation of end-to-end constraints into implementation parameters. Their method is confined to the single processor design of a real-time system. Later in [11], Manas et al. extend the approach for the design of a distributed real-time system, but they make a simplifying assumption that all messages in the system have a constant communication delay.

1.2. Organization

The rest of our paper is organized as follows. Section 2 presents the system design and implementation models, and gives an overview of our solution approach. Section 3 shows how intermediate constraints are derived. Section 4 presents the algorithms used to solve the intermediate constraints derived in Section 3. Finally, in Section 5 we conclude the paper with remarks on our experience and future research directions.

2. System design model versus implementation model

The approach that we present in this paper has been motivated largely by distributed real-time control systems, and much of the terminology derives from there [12]. In this section, we present the system and network model of the distributed real-time system. Then, we define the system design model which includes a task graph and associated end-to-end timing requirements. We also present the implementation model which possesses a set of periodic tasks and task specific attributes. Finally, we summarize our problem and show the overview of the solution approach.

2.1. System and network model

In many distributed real-time control systems, sensors are polled periodically for external inputs which is then processed by one or more controller tasks to generate commands for the actuators. While simple control systems may consist of a single such control loop, more and more systems are being built which consist of many interacting control loops executing at different rates sharing tasks and data. Furthermore, the tasks execute concurrently and may be physically distributed over a number of processing hosts.

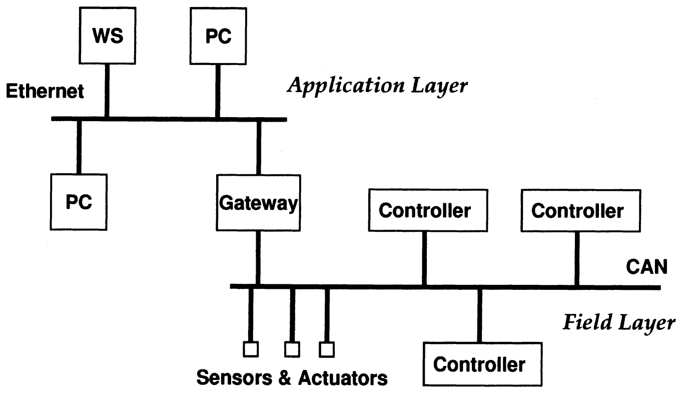


Fig. 2. Layered architecture of a distributed control system.

Fig. 2 shows a typical configuration for a distributed real-time control system. The system has two layers and they are connected through a gateway station. The eventual goal of this paper is in the design of the field layer.

The field layer connects sensors and actuators to processing hosts via the controller area network (CAN) [13] which supports real-time communication. Each processing host at the field layer is a single CPU controller. The sensors and actuators are autonomous devices connected to the CAN. We assume that the hosts are synchronized with respect to a global time base, and all our analyses are done with respect to this time base.

Each processing host has a suitable real-time operating system which can be used to implement periodic real-time tasks and perform schedulability analysis on it. Likewise, the CAN has a priority-based bus arbitration scheme which allows for schedulability analysis of periodic real-time messages [13].

2.2. Design model

2.2.1. Task graph

While many design models and methods exist in the literature [7], we use a simple messaging model to represent a real-time system. Such a model is described by a task graph which is defined as a directed graph $G(V, E)$ such that

- $V = \mathscr{P} \cup \mathscr{B}$ is a set of nodes where $\mathscr{P} = \{\tau_1, \ldots, \tau_n\}$ is a set of tasks and $\mathscr{B} = \{\pi_1, \ldots, \pi_m\}$ is a set of communicating ports,
- $E \subset (\mathscr{P} \times \mathscr{B}) \cup (\mathscr{B} \times \mathscr{P})$ is a set of directed edges between tasks and ports. $\tau_i \rightarrow \pi_j$ denotes task τ_i's write access to port π_j and $\pi_j \rightarrow \tau_i$ is τ_i's read access to π_j.

Fig. 3 gives an example of a task graph.

The task graph model incorporates many essential features such as task sharing, simple synchronization, software reusability, and network transparent communication. Several researchers have proposed similar models for distributed

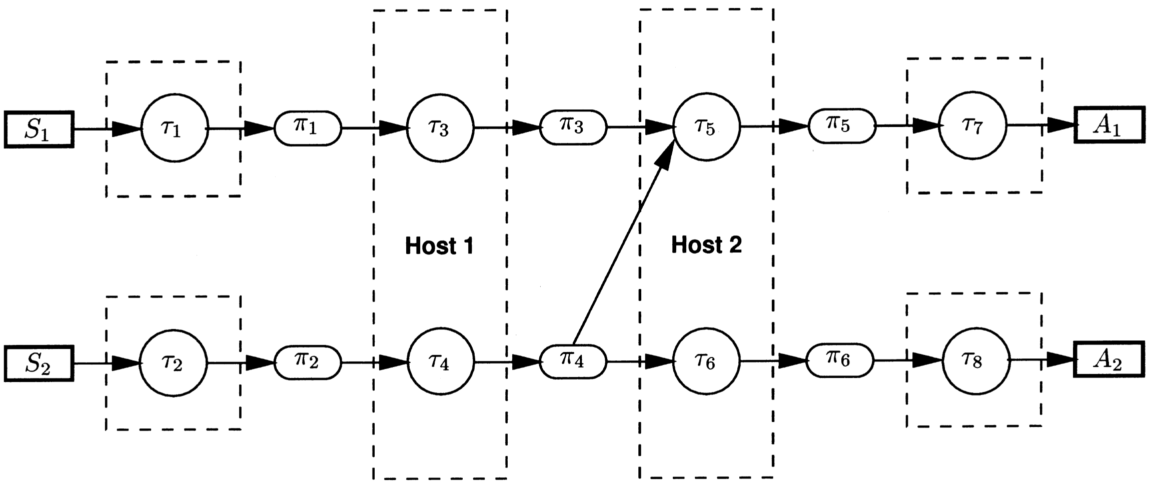


Fig. 3. Task graph design with task allocation.

control systems [14,15]. The properties of the task graph model are as follows:

1. A real-time system is composed of a set of communicating tasks. Tasks perform the computational activities in the system which consist of reading data from all its input ports, operating on the data, and finally, writing data to its output ports.
2. Tasks communicate with each other through ports. Ports provide a network transparent message passing abstraction. Each port has a single writer restriction, but may have an arbitrary number of readers. The ports are accessed by the tasks through generic operations such as "read" and "write". The code to implement these operations is instantiated automatically when the system is configured.
3. The writes to a port are always asynchronous and non-blocking. On the other hand, the reads are synchronous, that is a process reading from a port waits for data to be written. Synchronous communication reduces end-to-end latency and jitter. Thus, each communication establishes a precedence between the writer and the reader.
4. Sensors and actuators form the external interface of the system with the environment. We assume that each sensor is read by a special task, which copies the data from the sensor to a port to be read by computation tasks. In reality, this is done autonomously by the hardware logic of a sensor device. Likewise, there is an actuator task for each actuator which reads data from an input port and sends them to the actuator. As a sensor or an actuator is an autonomous device, it is treated as a special processor with a single sensor or actuator task.
5. A task is periodically activated at the start of the period, and for each activation it executes one iteration loop.

A real-time system designed with this model is thus represented as a finite, directed, acyclic graph where the tasks and the ports form the nodes in the graph, and the edges correspond to reads and writes to the ports. In such a design the path from a sensor to an actuator forms a chain of producer/consumer pairs forming an end-to-end computation. Timing constraints are often defined on such end-to-end computations. Thus, we use the term *transaction* for end-to-end computations which is defined as a relation between sensors and actuators. Let $\Gamma(\mathscr{S}\|\mathscr{A})$ represent a transaction that takes inputs from sensors in set $\mathscr{S}$, and produces outputs for actuators in set $\mathscr{A}$. The transaction then consists of all the tasks that fall on the path from any sensor in $\mathscr{S}$ to any actuator in $\mathscr{A}$.

2.2.2. Task allocation

Structurally, a task graph may be viewed as a set of disjoint subgraphs if we remove the nodes corresponding to network ports (and all edges incident on those nodes). We call each such disjoint subgraph a *partition*. In Fig. 3, $\langle\tau_3,\tau_4\rangle$ and $\langle\tau_5,\tau_6\rangle$ denote two such partitions. Each partition is mapped to a single host. The mapping of partitions to hosts defines the allocation of tasks, and we assume that this mapping is already defined. Such task allocation is static, and it does not change at run-time.

2.2.3. End-to-end timing constraints

The following types of timing constraints are allowed to be established on the transactions.

1. *Maximum Allowable Validity Time* (MAVT): This constraint ensures that a data sample is used before it loses its freshness, and thus, it bounds the maximum end-to-end delay permissible between the reading of a sensor and the delivery of the output command to an actuator based on that reading. We use the notation $M(S\|A)$ to represent this maximum delay from sensor S to actuator A.

2. *Input Data Synchronization*: This constraint ensures that when multiple sensors collaborate in driving an actuator, then the maximum time-skew between the sensor readings is bounded. We use the notation $Sync(S_1, S_2 \| A)$ to denote the maximum time-skew between two sensors S_1 and S_2. Let t_1 and t_2 be the time points when two sensors S_1 and S_2 are read to drive A, respectively. Then we have

$$|t_1 - t_2| \leqslant Sync(S_1, S_2 \| A).$$

3. *Maximum Transaction Period* (MTP): This constraint bounds the maximum period for an end-to-end computation. We use the notation $MaxP(S \| A)$ to denote the maximum period.

2.3. Implementation model

We assume that the real-time system is implemented as a set of periodic tasks. A periodic task τ_i is represented by a 5-tuple $\langle e_i, T_i, d_i, \phi_i, \mathscr{P}_i \rangle$, where e_i represents the task's execution delay, T_i its period, d_i its deadline relative to the start of period, ϕ_i its initial phase (denoting the initial activation time of the periodic task), and $\mathscr{P}_i$ the processor to which it is allocated. Fig. 4 depicts the timing attributes of a periodic task through a sample execution scenario of its first two activations.

The ports are implemented differently depending on whether the tasks using the port are on the same processor or not. When all the tasks using the port are allocated on the same processor, then the port is implemented simply as a message buffer to which writes and reads occur. On the other hand, when the tasks are allocated on different processors, the data must be transferred via the communication network to remote processors. In this case, the implementation consists of a set of message buffers, one at the sender's host and one at each receiver's host. Data is transferred from the sender's buffer to the receiver's buffer by a virtual communication task. On the receiver's side, the data retrieved from the network is directly placed into the receiver's buffer.

Data transfer over the communication network generates a set of periodic message streams. We label the message stream produced by task τ_i as m_i. The message stream m_i is treated analogously with a task with notation $\langle e_i^m, T_i^m, d_i^m, \phi_i^m, \mathscr{P}_i^m \rangle$ denoting the maximum message transmission time, the period, the deadline, the initial phase, and the

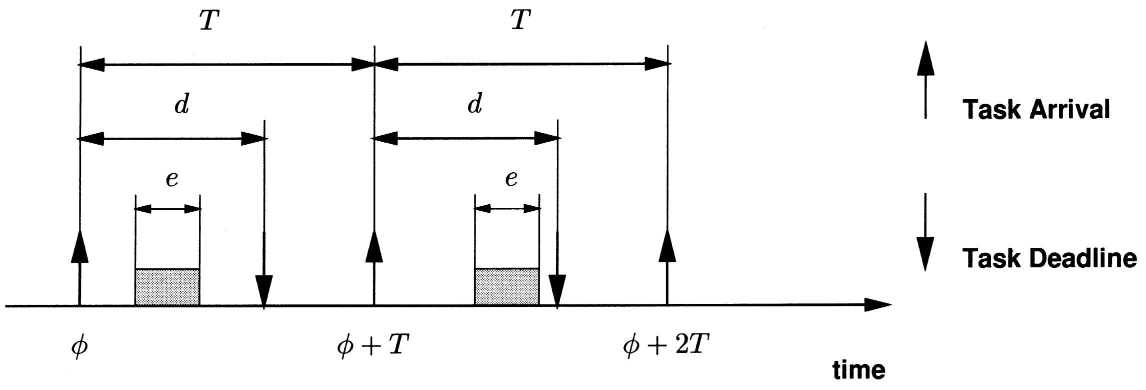


Fig. 4. Timing attributes of a periodic task.

network, respectively. In our analysis, the network is treated just like a processor.

2.4. A walk-through example

Fig. 3 shows the task graph for a simple real-time system with each task being allocated on a host. The system consists of two transactions, one driving actuator A_1 and the other driving actuator A_2. There are two sensors in the system S_1 and S_2 which are read by sensor tasks τ_1 and τ_2. Likewise, the actuators are given commands by actuator tasks τ_7 and τ_8. There are four control tasks allocated on two processors. All communications are synchronous. This example, which we use throughout the paper, helps us illustrate the key aspects of our methodology. The end-to-end constraints postulated on the system are given in Table 1.

Task execution delays are also specified in the table; we assume the following execution times: $e_1 = e_2 = e_7 = e_8 = 0$, i.e., the sensor and actuator tasks take negligible execution delay since they only involve simple copy operations performed by hardware independent of the host processors.

2.5. Problem description and solution overview

For a given task graph design of a system (Fig. 3), the end-to-end timing requirements (Table 1), and the task allocation (Fig. 3), our problem is to derive task specific attributes (i.e., the period, deadline, and phase) for each task and message stream. We model this problem as a constraint solving problem in which the end-to-end requirements are first expressed as a set of intermediate constraints on the task attributes. The intermediate constraints are then solved such that the results preserve the timing correctness, i.e., if the final task set (which is a solution of the constraints) is schedulable, then the original end-to-end requirements will be satisfied. Though we do not address arbitrarily complex schedulability analyses of the final task set, we do incorporate a utilization-based schedulability analysis into our constraint solving process, so that the solution derived is not trivially unschedulable. On the other hand, we treat network tasks differently from previous works [2,3] where message communication delays are considered to be constant and network utilization is not taken into account. Our approach is to represent network traffic in terms of network utilization, and decrease message transfer delays during the network schedulability analysis; hence we name the approach “network conscience”.

Generally, however, solving such constraints is not an easy problem due to several factors. First, in a distributed system there may be many tasks, and that may induce many variables. Second, the intermediate constraints are not always linear, as

Table 1
End-to-end timing constraints of the example system

Transactions	$\Gamma(S_1, S_2 \| A_1)$	$\Gamma(S_2 \| A_2)$
Input Data Synchronization	$\mathrm{Sync}(S_1, S_2 \| A_1) = 1$ ms	
Maximum Allowable Validity Time	$M(S_1, S_2 \| A_1) = 60$ ms	$M(S_2 \| A_2) = 80$ ms
Maximum Transaction Period	$\mathrm{MaxP}(S_1, S_2 \| A_1) = 20$ ms	$\mathrm{MaxP}(S_2 \| A_2) = 50$ ms
Maximum Task Execution Delay	$e_1 = 0$ ms, $e_2 = 0$ ms, $e_5 = 9$ ms, $e_6 = 15$ ms,	$e_3 = 7$ ms, $e_4 = 8$ ms, $e_7 = 0$ ms, $e_8 = 0$ ms

will become clear in Section 3. Third, incorporating schedulability into the constraint solving process significantly complicates the problem.

One way to tackle all this complexity is to use optimization techniques such as genetic algorithms or simulated annealing [16]. These approaches, however, have a major common drawback in that they do not give any feedback when a solution cannot be found. Therefore, we take a different approach that is based on decomposing the constraint solving problem into a sequence of subproblems. The motivation behind this approach is that for each subproblem, we can use specialized heuristics and performance metrics. This will allow us to analyze the system more easily, thereby providing useful feedback upon failure. However, the success of the overall approach depends critically on whether the constraint solving problem can be decomposed into well-defined subproblems. For our case, we decompose this problem into the *Period Assignment* and the *Phase and Deadline Assignment* subproblems which are outlined below.

1. *Period Assignment*: The first subproblem is to assign periods to tasks with an objective of minimizing the utilization of processors. Since the utilization depends only on periods and execution times of tasks, it can be computed without having knowledge of any other task attribute.
2. *Phase and Deadline Assignment*: Once the periods are determined, we proceed to determine the phase and deadline of each task. The main problem here is to determine a set of individual message deadlines such that we can find some way to schedule each message within its deadlines. The deadline of a message is chosen from an interval between its worst-case transmission delay and its period. The phases are determined last and any solution consistent with the constraints is acceptable. As phases are used to maintain precedence among tasks and input data synchronization, we do not associate any schedulability measure with them.

It is essential that the two subproblems are solved in the given order as it is very difficult to obtain useful schedulability criteria the other way round. We defer more precise specifications of the subproblems until later after we have shown how the entire constraint problem is set up.

3. Deriving the intermediate constraints

The first step in our methodology is to transform the end-to-end constraints and synchronization requirements into a set of intermediate constraints on task attributes. This is a three-step process which is represented by the following three subsections.

3.1. Intermediate constraints from producer/consumer model

The producer/consumer model forms the basic communication semantics in our transaction model. In fact, a transaction can be redefined as a series of producer/consumer pairs. A producer/consumer pair inherently incurs blocking synchronization in that a consumer task must wait for a producer task to generate the needed data. In the periodic task model, this also induces dependencies between task periods which may result in unnecessarily high rates of execution for some tasks. Consider, for example a producer task τ_{p} writing to a port π which is read synchronously by two consumer tasks τ_{c_1} and τ_{c_2}. To meet the synchronization requirement of the producer/consumer, each execution of the consumer must wait for fresh data to be written. This normally requires that the activation rate of the consumer be the same as that of the producer, i.e., $T_{c_1} = T_{c_2} = T_{\mathrm{p}}$.

Consider a situation where τ_{c_1} and τ_{c_2} have maximum periods of 100 ms and 350 ms, respectively. Due to the synchronization requirement τ_{c_1}, τ_{c_2}, and τ_{p} all would need to execute at a period of 100 ms. However, this high rate is unnecessary for τ_{c_2} which would result in wasted processor capacity. Furthermore, this may result in the system being unschedulable. A better solution might be to set $T_{\mathrm{p}} = T_{c_1} = 100$ and $T_{c_2} = 300$. With this solution, we have less waste of processor capacity while still providing the following clean semantics for synchronization.

Task τ_{c_2} synchronizes with every third execution of τ_{p}.

Such synchronization is satisfied only when the consumer's period is an integral multiple of the producer's period, and we refer this relation to be harmonic. Therefore, to enforce this synchronization, we impose a harmonicity constraint between any producer/consumer pair. This is represented as $T_{\mathrm{p}}|T_{\mathrm{c}}$, where the operator $|$ is interpreted as "exactly divides". Given a task graph for a system, the harmonicity constraints can be represented in a harmonicity graph in which the task periods form the nodes and the edges represent the harmonicity constraints. That is, an edge $T_i \rightarrow T_j$ represents the constraint $T_i|T_j$. Fig. 5 shows the harmonic graph of our walk-through example.

3.2. Intermediate constraints from precedence and end-to-end delay

One of the end-to-end properties of interest to us is the end-to-end delay from a sensor reading to an actuator output. Since data flow through the tasks on the path from the sensor to the actuator and delay is incurred at each step computation of the intermediate delays is required. This flow of data is a producer/consumer model, and recall that under producer/consumer synchronization, a precedence relation between the producer and consumer is required. Otherwise, the consumer task might be blocked until the producer generates the needed data. This complicates the scheduling of the consumer task since its execution is controlled by another task running on a different host.

A simple way to enforce this precedence is through the use of task phase variables. For a given producer/consumer pair $(\tau_{\mathrm{p}}, \tau_{\mathrm{c}})$ and message task τ_{p}^m, we have the following constraints:

$$\phi_{\mathrm{p}}^m \geqslant \phi_{\mathrm{p}} + d_{\mathrm{p}} \text{ and } \phi_{\mathrm{c}} \geqslant \phi_{\mathrm{p}}^m + d_{\mathrm{p}}^m.$$

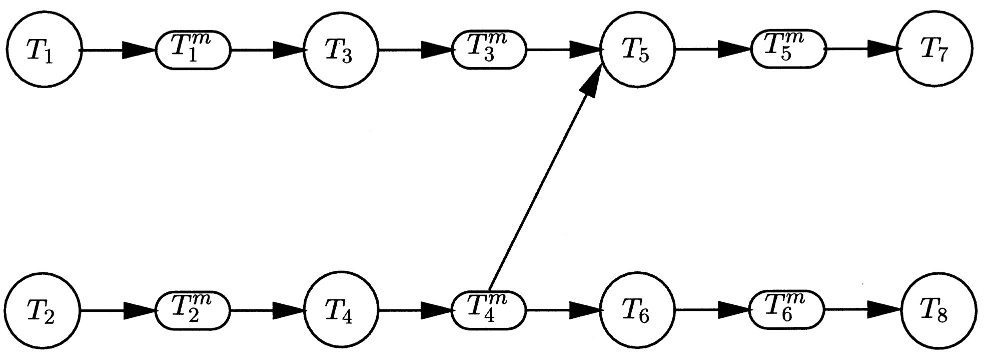


Fig. 5. The harmonicity graph.

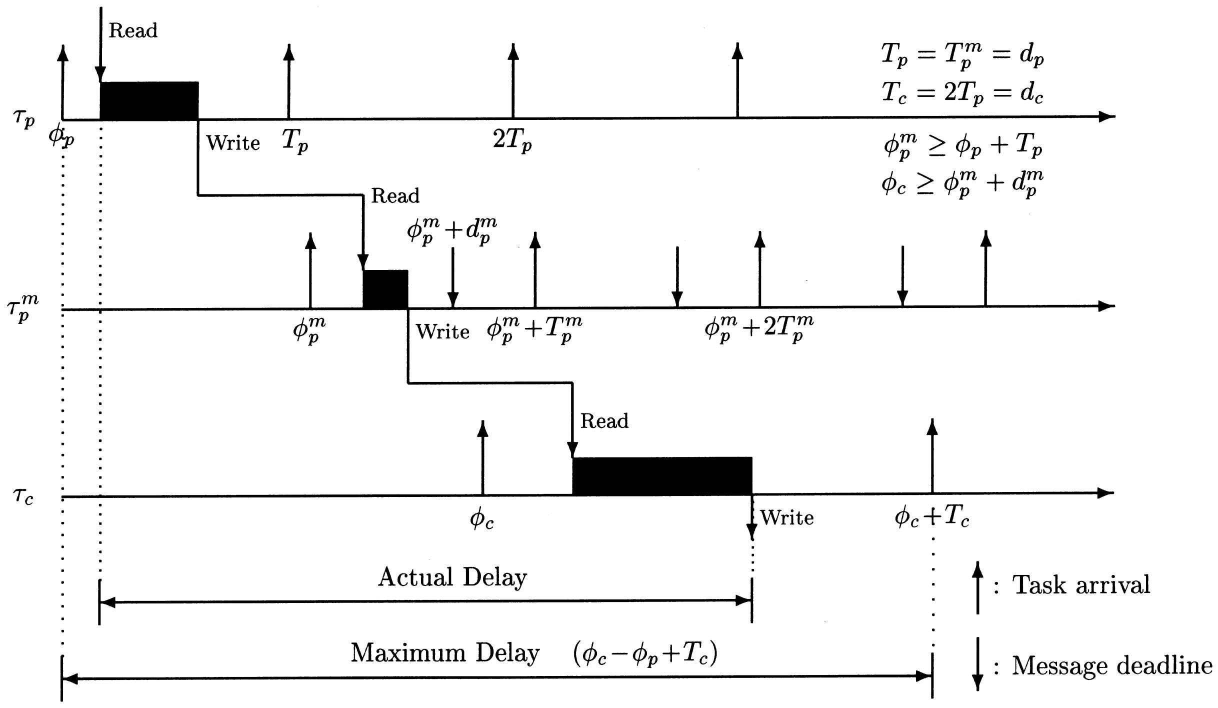


Fig. 6. Delay and precedence in communication.

Note that such constraints must be added for each producer/consumer pair.

When $T_c = kT_p$, we require the following precedence relation: the ith iteration of task τ_c reads data produced by the kith iteration of task τ_p (assuming the iteration numbers start from 0). The phase difference between the corresponding iterations is then $\phi_c - \phi_p$ the initial phase difference.

Fig. 6 illustrates such synchronous data communication when $T_c = 2T_p$, $T_p = d_p$ and $T_c = d_c$. A data transfer from the producer τ_p to the consumer τ_c using port π is shown on the time lines. The message task τ_p^m is shown on the second time line. We are interested in finding the maximum delay from the time τ_p reads data from its input ports to the time τ_c writes data to its output ports. The worst case is when τ_p reads its input data just as it is invoked and τ_c writes to the output ports at the end of its period. Therefore, the worst-case delay becomes $\phi_c - \phi_p + T_c$. By extending the same logic to a chain of producer/consumer tasks, we can derive the end-to-end delay from a sensor task τ_s to an actuator task τ_a as $\phi_a - \phi_s + d_a$. Thus, for each validity time requirement $M(S\|A)$, we have the following intermediate constraint:

$$\phi_a - \phi_s + d_a \leqslant M(S\|A).$$

In summary, for each end-to-end delay constraint between a sensor and an actuator, we have one constraint to satisfy the delay bound, and a

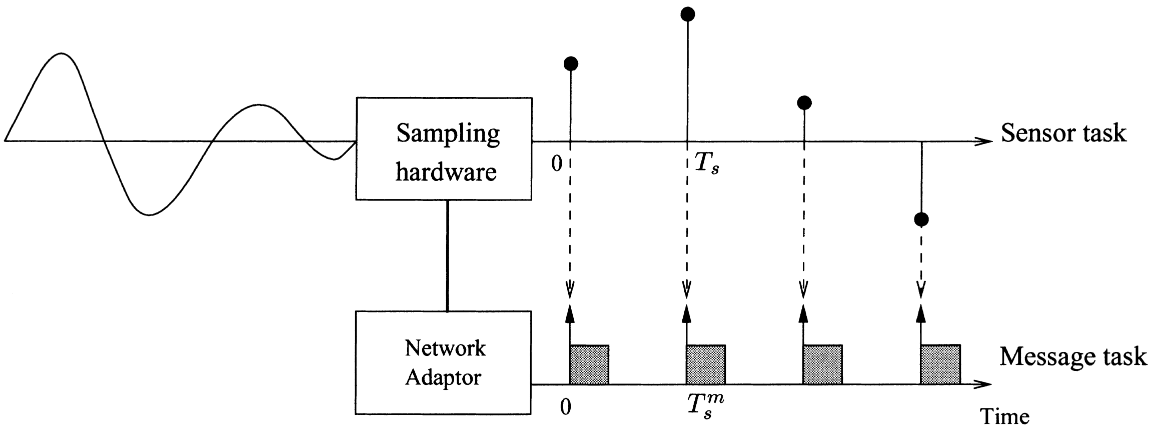


Fig. 7. A synchronized sensor model.

number of precedence constraints for each producer/consumer task pair along the path from the sensor to the actuator. Fortunately, many of these constraints may be replaced by equalities, thereby reducing the number of free variables and constraints to be solved. A notable simplification occurs when a task has a single input port and, therefore, must wait for only one producer. In this case, we can set the phase of the consumer to coincide with the deadline of the message task, i.e., $\phi_c = \phi_p^m + d_p^m$, thereby eliminating the variable ϕ_c from the constraint set.

3.3. Intermediate constraints from input data synchronization

A variety of sensing devices such as switches, counters, and analog to digital (A/D) converters are used at the field layer. Each of these devices usually consists of two functional modules: a hardware module executing the virtual sensor task and a network adaptor module executing the virtual message task. Fig. 7 pictorially depicts such a sensor model using an A/D converter which converts analog signals to digitized values. The sensor task (producer) produces the sensor readings which are delivered to the control tasks (consumers) by the message task. Since a sensor always works as a producer for consumer τ_c and is attached directly on a network, we have $T_s|T_s^m$ and $T_s^m|T_c$. As we assume that the execution time of the sensor task to be $e_s = 0$, the deadline of the sensor task can be set $d_s = 0$ which means that a sensor reading is available at the beginning of every period of the sensor task. From the sensor task's point of view, the only reader of the sensor readings is the message task. Thus, the period of the sensor task can be set equal to that of the message task, that is $T_s = T_s^m$.

Now, consider an input data synchronization requirement $Sync(S_1, S_2 \| C_1)$ where the sensor inputs S_1 and S_2 must be synchronized. Let τ_{s_1} and τ_{s_2} be the sensor task reading from S_1 and S_2, respectively. Then, the corresponding message tasks are $\tau_{s_1}^m$ and $\tau_{s_2}^m$. Fig. 8 depicts an example of the sensor data synchronization and the communication required between the sensors and the consumers.

Since the activation of the sensor tasks should be made at the beginning of any transaction, the

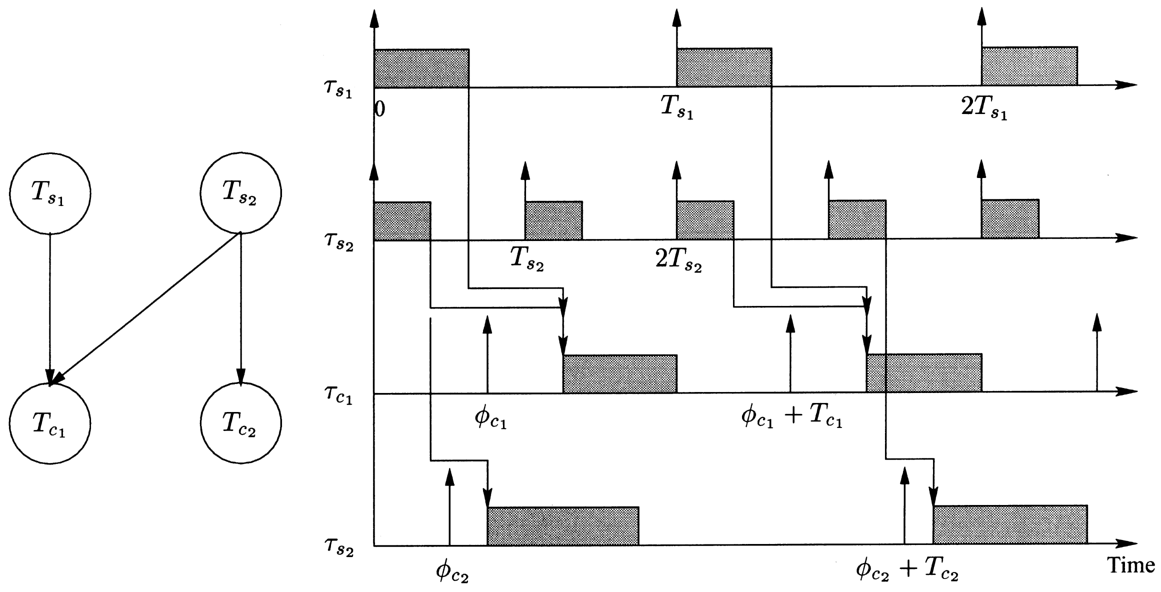


Fig. 8. Input data synchronization between two sensors.

initial phases of all the sensors should be equally set to zero. The message tasks of the sensors can also start at the activation time of the sensors as shown in Fig. 7. That is,

$$\phi_{s_1} = \phi^m_{s_1} = \phi_{s_2} = \phi^m_{s_2} = 0.$$

Refer to the two sensor example in Fig. 8. Harmonicity constraints are imposed between the producer and the consumer: $T^m_{s_1}|T_{c_1}$, $T^m_{s_2}|T_{c_1}$, and $T^m_{s_2}|T_{c_2}$. From the harmonicity constraints, T_{c_1} is the least common multiple (LCM) of $T^m_{s_1}$ and $T^m_{s_2}$, and thus consumer task τ_{c_1} uses the sensor readings which are sampled at every LCM times of $T^m_{s_1}$ and $T^m_{s_2}$. This shows that the input data synchronization requirement is satisfied if the harmonicity constraints imposed between the producers (sensors) and consumers are satisfied.

3.4. Deriving the intermediate constraints for the walk-through example

Let us now return to the walk-through example and derive the intermediate constraints for the tasks. We divide the constraints into two sets: one for periods and one for phases and deadlines.

Constraints on periods: The harmonicity constraints can easily be derived from the harmonicity graph as shown in Fig. 5. We can replace the harmonicity constraints of certain producer/consumer pairs with equalities if the consumer reads input from only one producer and the producer has only one consumer. We have the following equalities.

$$T_1 = T^m_1 = T_3 = T^m_3, \qquad T_2 = T^m_2 = T_4 = T^m_4,$$

$$T_5 = T^m_5 = T_7, \qquad T_6 = T^m_6 = T_8.$$

The final set of harmonicity constraints are $T_3|T_5$, $T_4|T_5$ and $T_4|T_6$. We derive the lower and upper bounds of the task periods from the maximum execution delays of the tasks and the period requirements $\mathrm{MaxP}(S_1, S_2\|A_1) = 20$ and MaxP $(S_2\|A_2) = 50$, respectively. Thus, we have

$$7 \leqslant T_3 \leqslant 20, \quad 8 \leqslant T_4 \leqslant 20, \quad 9 \leqslant T_5 \leqslant 20, \quad 15 \leqslant T_6 \leqslant 50.$$

Constraints on phases and deadlines: From the phasing and input data synchronization, the initial phases of the sensor tasks and the message tasks of sensors are

$$\phi_1 = \phi_1^m = \phi_2 = \phi_2^m = 0.$$

The maximum allowable validity time requirements $M(S_1, S_2\|A_1) = 60$ and $M(S_2\|A_2) = 80$ impose the following constraints:

$$\begin{aligned}
&\phi_7 - \phi_1 + d_7 = \phi_7 + d_7 \leqslant 60,\\
&\phi_7 - \phi_2 + d_7 = \phi_7 + d_7 \leqslant 60,\\
&\phi_8 - \phi_2 + d_8 = \phi_8 + d_8 \leqslant 80.
\end{aligned}$$

As pointed out earlier many of the phase and deadline variables can be removed through equalities. The following steps show the effect of such simplifications on the constraints.

1. Since the actuators as well as the sensors are autonomous devices and the execution times of the sensor and actuator tasks are negligible, we can set $d_1 = d_2 = d_7 = d_8 = 0$.
2. For each task which has a single input port, we set its phase to match the deadline of its producer task. Thus, we get

$$\begin{aligned}
\phi_3 &= \phi_1^m + d_1^m &&= d_1^m,\\
\phi_4 &= \phi_2^m + d_2^m &&= d_2^m,\\
\phi_6 &= \phi_4 + d_4 + d_4^m &&= d_4 + d_2^m + d_4^m,\\
\phi_7 &= \phi_5 + d_5 + d_5^m,\\
\phi_8 &= \phi_6 + d_6 + d_6^m &&= d_4 + d_6 + d_2^m + d_4^m + d_6^m.
\end{aligned}$$

3. Since task τ_5 requires data from both τ_3 and τ_4, the phase constraints of τ_5 are

$$\phi_5 \geqslant d_3 + d_1^m + d_3^m \quad \text{and} \quad \phi_5 \geqslant d_4 + d_2^m + d_4^m.$$

4. By substituting the values in the maximum allowable validity time constraints, the following constraints are obtained:

$$\phi_5 + d_5 + d_5^m \leqslant 60 \text{ and } d_4 + d_6 + d_2^m + d_4^m + d_6^m \leqslant 80.$$

5. For simplicity, we let $d_i = T_i$ for all control tasks. This does not seriously restrict our approach since such an assumption is frequently made in many practical real-time systems for the purpose of efficient schedulability analysis. If we were to allow control tasks to have different values for the deadlines and periods, we would first have to determine the task priorities which is another complex problem. On the other hand, we still allow different deadlines and periods for the network tasks (messages) whose priorities can be determined according to a well-defined rule of message importance [13].

In summary, the final set of constraints on phases and deadlines are

$$\begin{aligned}
&\phi_3 = d_1^m, && \phi_4 = d_2^m\\
&\phi_6 = d_4 + d_2^m + d_4^m && \phi_7 = \phi_5 + d_5 + d_5^m,\\
&\phi_8 = d_4 + d_6 + d_2^m + d_4^m + d_6^m, && \phi_5 \geqslant d_3 + d_1^m + d_3^m,\\
&60 \geqslant \phi_5 + d_5 + d_5^m, && \phi_5 \geqslant d_4 + d_2^m + d_4^m,\\
&80 \geqslant d_4 + d_6 + d_2^m + d_4^m + d_6^m.
\end{aligned}$$

The entire set of the derived intermediate constraints are given in Table 2.

4. Solving the intermediate constraints

Once the intermediate constraints have been derived, we are left with a set of tasks with constraints on their periods, phases, and deadlines. These, for the walk-through example, were shown in Table 2. Using these constraints, we are now in a position to assign the periods, phases, and

Table 2
The whole set of derived intermediate constraints

Harmonicity requirement	$T_3 \mid T_5,\ T_4 \mid T_5,\ T_4 \mid T_6$
Equal periods	$T_1 = T_1^m = T_3 = T_3^m$ $T_2 = T_2^m = T_4 = T_4^m$ $T_5 = T_5^m = T_7$ $T_6 = T_6^m = T_8$
Period ranges	$7 \leqslant T_3 \leqslant 20,\ 8 \leqslant T_4 \leqslant 20,\ 9 \leqslant T_5 \leqslant 20,\ 15 \leqslant T_6 \leqslant 50$
Precedence and delay	$\phi_3 = d_1^m$ $\phi_4 = d_2^m$ $\phi_6 = d_4 + d_2^m + d_4^m$ $\phi_7 = \phi_5 + d_5 + d_5^m$ $\phi_8 = d_4 + d_6 + d_2^m + d_4^m + d_6^m$ $\phi_5 \geqslant d_3 + d_1^m + d_3^m$ $\phi_5 \geqslant d_4 + d_2^m + d_4^m$ $60 \geqslant \phi_5 + d_5 + d_5^m$ $80 \geqslant d_4 + d_6 + d_2^m + d_4^m + d_6^m$
Partial deadline assignment	$d_1 = d_2 = d_7 = d_8 = 0\ \forall i\ 3 \leqslant i \leqslant 6,\ d_i = T_i$

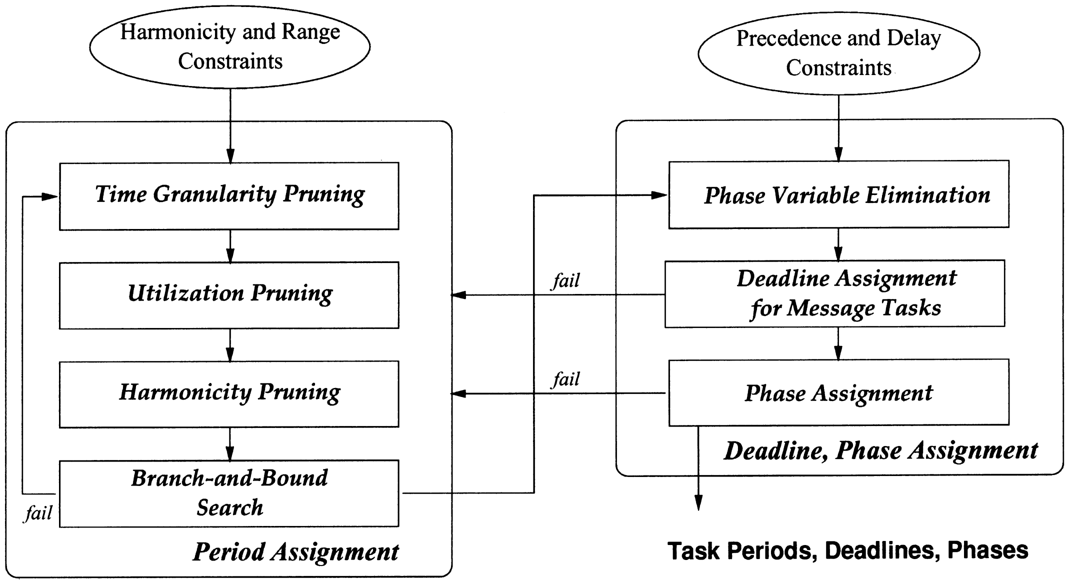

Harmonicity and Range Constraints
Time Granularity Pruning
Utilization Pruning
Harmonicity Pruning
Branch-and-Bound Search
fail
Period Assignment
Precedence and Delay Constraints
Phase Variable Elimination
Deadline Assignment for Message Tasks
Phase Assignment
fail
fail
Deadline, Phase Assignment
Task Periods, Deadlines, Phases

Fig. 9. The intermediate constraint solving procedure.

deadlines to the tasks. Fig. 9 depicts the procedure for assigning these values. As shown in the figure, the whole process is an iterative one, and can be divided into two components, that is, the period assignment component and the deadline and phase assignment component.

The period assignment component accepts the harmonicity and the range constraints as input and generates task periods with the objective of minimizing processor utilization. This is not an easy problem as it involves nonlinear integer programming due to the harmonicity requirements. To attack this problem, we convert the original optimization problem into a decision problem through the use of *cut-off* utilizations which are provided by the user. Our approach to the period assignment problem, then, is to perform a branch-and-bound search after we reduce the search space of the problem as much as possible. To reduce the search space, we use three pruning algorithms, namely, *Time Granularity Pruning*, *Utilization Pruning*, and *Harmonicity Pruning* which are explained in the following subsection.

Once the periods have been assigned, the deadline and phase assignment component takes these results and the precedence and delay constraints as input to obtain the deadline and phase values. This component is subsequently composed of the *Phase Variable Elimination*, *Deadline Assignment for Message Tasks*, and *Phase Assignment* steps. If any of these steps fails, we return to the period assignment component to obtain a new period assignment.

We elaborate on each of these two components and their respective steps in the following two subsections.

4.1. Period assignment

As mentioned above, this component consists of the pruning and search steps. These steps are illustrated using the walk-through example. Again, we ignore the sensor and actuator tasks since they do not involve the host processors, and hence, we are left with four tasks τ_3, τ_4, τ_5, and τ_6 allocated on two processors. From Table 2 the initial feasible range for each period is $T_3 \in [7, 20]$, $T_4 \in [8, 20]$, $T_5 \in [9, 20]$, and $T_6 \in [15, 50]$. Starting from these values the solution proceeds in the following steps. Each step is depicted in Fig. 10.

1. *Time Granularity Pruning*: We require that the period of each task be an integral multiple of a given time granularity g_r. g_r has the characteristic that the larger the value, the smaller the search space becomes; however, this decreases the chances of finding a feasible solution. In order to minimize the search space initially, we start out with a large g_r. If this fails to yield a solution, we restart the process with a smaller value. Selecting a right initial value for g_r is problem specific making its choice difficult to generalize. However, as our solution finding process eventually leads to a selection that yields a feasible solution, if one exists, the initial choice is really insignificant in terms of correctness. For our example, we choose a time granularity of 5. This reduces the original period ranges into the following sets of values:

$$T_3 \in \{10, 15, 20\}, \quad T_4 \in \{10, 15, 20\},$$

$$T_5 \in \{10, 15, 20\},$$

$$T_6 \in \{15, 20, 25, 30, 35, 40, 45, 50\}.$$

2. *Utilization Pruning*: In this step, we use the cut-off utilization for each host to tighten the lower bound of the periods. For a set $\mathscr{T}^k$ of tasks allocated on host $\mathscr{P}^k$, the processor utilization U^k is defined as

$$U^k = \sum_{\tau_i \in \mathscr{T}^k} \frac{e_i}{T_i},$$

where $\mathscr{T}^k = \{\tau_1, \tau_2, \ldots, \tau_m\}$ and T_i is the period assigned τ_i. In order for $\{T_1, T_2, \ldots, T_m\}$ to be a feasible assignment, U^k must be no greater than

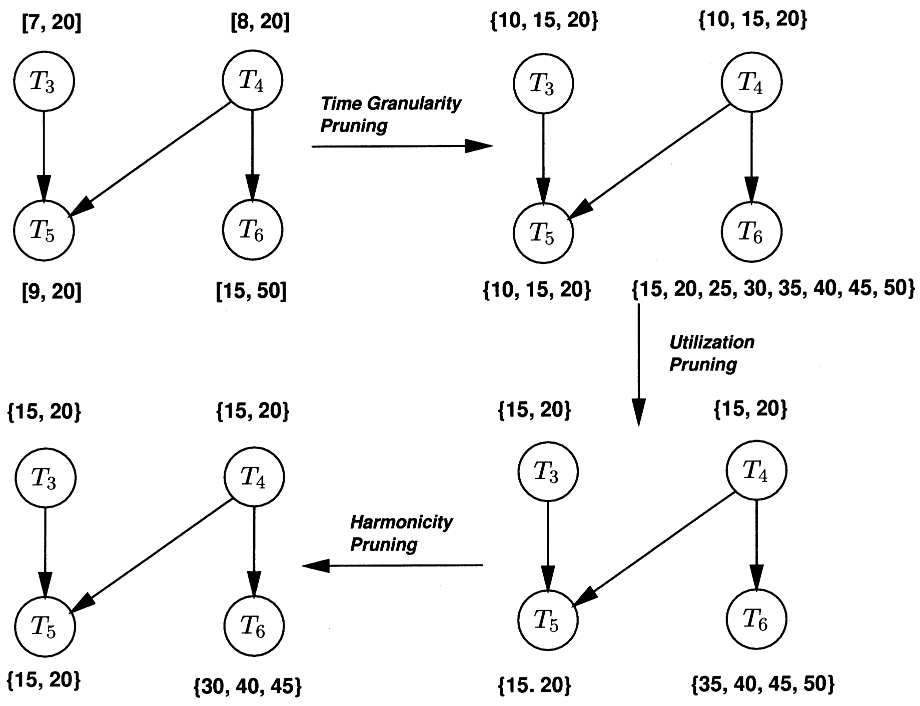


Fig. 10. Period assignment: feasible range reduction.

the cut-off utilization U_c^k of host $\mathscr{P}^k$. Thus we have the following relation.

$$\sum_{\tau_i \in \mathscr{T}^k} \frac{e_i}{T_i} \leqslant U_c^k \leqslant 1. \qquad (1)$$

To obtain a lower bound for the period of τ_i, we solve Eq. (1) by plugging in the largest period values for $\tau_j \in \mathscr{T}^k - \{\tau_i\}$. In our example, we use 0.9 as the cut-off utilization for each host. This implies that

$$U_c^1 = \frac{7}{T_3} + \frac{8}{T_4} \leqslant 0.9 \text{ and } U_c^2 = \frac{9}{T_5} + \frac{15}{T_6} \leqslant 0.9.$$

From this, to obtain the lower bound of T_4 we substitute T_3 with 20, the largest value among the periods. We get $7/20 + 8/T_4 \leqslant 0.9$, which gives $T_4 \geqslant 15$, after rounding to the nearest integer. Similarly, we get $T_3 \geqslant 14$, $T_5 \geqslant 15$, and $T_6 \geqslant 34$. Then, the feasible sets of values become $T_3 \in \{15, 20\}$, $T_4 \in \{15, 20\}$, $T_5 \in \{15, 20\}$, and $T_6 \in \{35, 40, 45, 50\}$.

3. *Harmonicity Pruning*: Harmonicity pruning is performed on the harmonicity graph based on the harmonicity relationship between the tasks. Recall that a node n_i in a harmonicity graph contains a set $T_{i,\text{set}}$ of period values. For all nodes n_j which are the immediate predecessors of n_i, the set $T_{j,\text{set}}$ is reduced as below.

$$T_{j,\text{set}} := T_{j,\text{set}} \cap \{a \in T_{j,\text{set}} | b \in T_{i,\text{set}} \text{ and } a|b\}.$$

Also, for all nodes n_j which are the immediate successors of n_i, the set $T_{j,\text{set}}$ is further reduced as below.

```
Algorithm Time Granularity Pruning
{
    input: $g_r$
    foreach task $\tau_i$ in transaction $\Gamma_j(S\|A)$
    {
        $T_{i,set} = \{t \mid t = g_r \cdot x, \lceil \frac{e_i}{g_r} \rceil \le x \le \lfloor \frac{MaxP_j}{g_r} \rfloor\}$
        if ($T_{i,set} == \{\}$) return fail;
    }
}
```

```
Algorithm Utilization Pruning
{
    input: $\{U_c^i :: P_i\}$
    foreach Host $P_i, \langle \tau_1, \tau_2, \cdots, \tau_l \rangle$
    {
        foreach task $\tau_j :: P_i$
        {
            $T_{j,set} = \{T \mid T \ge e_j \cdot (U_c^i - \sum_{k=1,\ k \ne j}^{l} \frac{e_k}{\max\{T_{k,set}\}})^{-1},\ T \in T_{j,set}\}$
            if ($T_{j,set} == \{\}$) return fail;
        }
    }
}
```

```
Algorithm Harmonicity Pruning
{
    input: $\{T_i | T_j\}$
    do
    {
        foreach harmonicity constraint $T_i | T_j$
        {
            $T_{j,set} = T_{j,set} \bigcap \{T \in T_{j,set} \mid T' \in T_{i,set}$ and $T' | T\}$
            if ($T_{j,set} == \{\}$) return fail;
            $T_{i,set} = T_{i,set} \bigcap \{T \in T_{i,set} \mid T' \in T_{j,set}$ and $T | T'\}$
            if ($T_{i,set} == \{\}$) return fail;
        }
    } while (change on period set)
}
```

Fig. 11. Three heuristic pruning algorithms.

$$T_{j,\mathrm{set}} := T_{j,\mathrm{set}} \cap \{a \in T_{j,\mathrm{set}} | b \in T_{i,\mathrm{set}} \text{ and } b|a\}.$$

This pruning is repeated until no further reduction is possible. In our example, we have a harmonicity constraint $T_4|T_6$. Thus, for T_6, the values 35 and 50 are eliminated since neither 15 nor 20 evenly divides them. Then, the feasible sets of values for the periods are

$$T_3 \in \{15,20\},\ T_4 \in \{15,20\},\ T_5 \in \{15,20\},\ \text{and}\ T_6 \in \{40,45\}.$$

The detailed algorithms for each of the three pruning steps are shown in Fig. 11

After the three pruning steps, we are left with a reduced search space on which the search is performed to find a feasible solution. A simple branch-and-bound heuristic may be used to control the search. In this paper, we make use of the harmonic period search algorithm which employs harmonicity and utilization checks as bounding functions [10]. As a result, we obtained the following period assignment.

	T_3	T_4	T_5	T_6	U^1	U^2	U^m
Solution	20	20	20	40	0.75	0.83	0.38

In the case that the search algorithm fails to find a solution, we return to the time granularity pruning step, this time choosing a smaller time granularity to enlarge the search space.

4.2. Phase and deadline assignment

Once the periods have been determined, we proceed to solve the precedence and delay constraints in Table 2 for the phase and deadline variables. This process is done in three steps as outlined below.

(1) *Phase Variable Elimination*: We begin by eliminating the phase variables from the precedence and delay constraint set using Fourier Variable Elimination [17]. Basically, this involves rewriting the constraints as lower and upper bound constraints on the variable to be eliminated, and then combining each lower bound with each upper bound. For our walk-through example, we have only one free phase variable, which is eliminated as shown below.

$$\begin{aligned}
&\phi_5 \geqslant d_3 + d_1^m + d_3^m\\
&\phi_5 \geqslant d_4 + d_2^m + d_4^m\\
&\phi_5 \leqslant 60 - d_5 + d_5^m\\
&d_4 + d_6 \leqslant 80 - d_2^m - d_4^m - d_6^m\\
&\overset{\text{Eliminate } \phi_5}{\Rightarrow} \quad d_3 + d_5 + d_1^m + d_3^m + d_5^m \leqslant 60\\
&\quad d_4 + d_5 + d_2^m + d_4^m + d_5^m \leqslant 60\\
&\quad d_4 + d_6 + d_2^m + d_4^m + d_6^m \leqslant 80.
\end{aligned}$$

(2) *Deadline Assignment*: At this stage, we have a system of linear constraints on the deadlines obtained from the previous step. We solve these constraints for the deadlines variables. As mentioned previously, while control tasks have deadlines equal to their periods, message tasks may have deadlines shorter than their periods. In order to effectively assign deadlines to message tasks, we take a network conscious approach and utilize a real-time communication network. The real-time communication network that we consider is the CAN [13]. Detailed descriptions of how deadlines are assigned to message tasks is deferred to Section 4.3, and for now, we simply state the solution obtained.

$$d_1^m = 3,\quad d_2^m = 3,\quad d_3^m = 5,\quad d_4^m = 6,\quad d_5^m = 7,\quad d_6^m = 7.$$

(3) *Phase Assignment*: Finally, once the deadlines have been assigned, we proceed to determine values for the task phases. This is done by plugging back the computed deadlines into the original precedence and delay constraints in Table 2. For

each phase variable, we assign the smallest value which satisfies the constraints. In our walk-through example, we have only one unsolved phase variable ϕ_5 which is assigned the value 29.

The values for the remaining phase and deadline variables are automatically assigned through the equalities derived in Section 3. The final task set parameters for the example system are shown below.

	e	ϕ	T	D		e	ϕ	T	D
τ_1	0	0	20	0	τ_2	0	0	20	0
τ_1^m	1	0	20	3	τ_2^m	1	0	20	3
τ_3	7	3	20	20	τ_4	8	3	20	20
τ_3^m	2	23	20	20	τ_4^m	2	23	20	6
τ_5	9	29	20	20	τ_6	15	29	40	40
τ_5^m	1	49	20	7	τ_6^m	1	69	40	7
τ_7	0	56	20	0	τ_8	0	76	40	0

4.3. Deadline assignment for message tasks

In this section, we show how message delivery deadlines are assigned to message tasks using the CAN. The reason for choosing this network is that it provides two benefits. First, it provides for a broadcast bus which is desirable for implementing multiple readers in a distributed real-time system. Second, it allows for simple mechanical arbitration solely relying on a static priority scheme. This, in turn, enables us to use a relatively simple schedulability test which allows us to predict the worst-case message transmission delays for a given set of priority-ordered, periodic messages. Our distributed system model is amenable for such analysis since all real-time messages in the system are periodic.

Our approach accepts a set of periodic real-time messages $\{m_1, m_2, \ldots, m_k\}$ which are ordered by the user assigned priorities. (The smaller index implies higher priority.) We then determine the lower bounds of message task deadlines using the fact that a message deadline must be no smaller than its worst-case message transmission delay. The mapping of messages onto priorities is essentially a function of the contents and importance of each message which is problem specific. Thus, we do not address this issue in this paper.

Instead of giving technical details of the CAN, we illustrate its communication protocol with an example. Fig. 12 shows the priority-based message transmission on the CAN. Initially, message m_{100} is ready and the CAN bus is idle. Thus, it is immediately transmitted. Before the completion of m_{100}, new messages m_{70} and m_{50} arrive. Though m_{70} arrived earlier than m_{50}, m_{50} begins transmitting next since it has higher priority than m_{70}. We see that message m_{70} is pushed back further since other higher priority messages m_{60} and m_{50} have arrived.

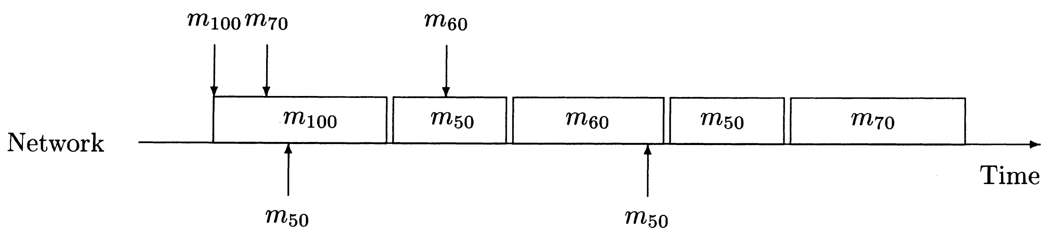


Fig. 12. Message priority and transmission oder.

Since message transmission on the CAN is nonpreemptible and totally priority-driven, it is easy to determine the worst-case transmission time of the highest-priority message, which is the sum of the transmission time of the largest CAN message and that of the highest-priority message itself. On the other hand, it is not trivial to generalize the computation of the worst-case transmission time of a message with an arbitrary priority. In [13], Burns et al. present an analytic technique which allows us to determine the worst-case transmission time of a message when the system possesses only periodic messages whose priorities are given. Using their results, the worst-case transmission time R_i^m of message m_i with priority i is written as follows:

$$R_i^m = B^m + W_i^m + C_i^m.$$

The term B^m represents the time taken to transmit the largest possible message which is already in transmission when m_i becomes ready. This value is reported to be 1.3 ms at 100 Kbits/s transmission speed for a maximum message size of eight bytes [13]. The term W_i^m is the queuing delay due to higher priority messages delaying the message m_i while it is in the queue. The last term C_i^m is the time taken to send the message m_i.

Let R_i^m be the worst-case transmission time of m_i. Taking into account multiple activations of higher priority messages during the time R_i^m, we rewrite the worst-case queuing delay as below.

$$W_i^m = \sum_{\forall j \in hp(m_i)} \left\lceil \frac{R_i^m}{T_j^m} \right\rceil c_j^m,$$

where $hp(m_i)$ denotes the set of messages which have higher priority than m_i. Then the worst-case transmission time of m_i is given as below.

$$R_i^m = B^m + \sum_{\forall j \in hp(m_i)} \left\lceil \frac{R_i^m}{T_j^m} \right\rceil c_j^m + c_i^m. \tag{2}$$

As Eq. (2) is a recurrence equation on R_i^m, an iterative algorithm can compute R_i^m by initially assigning it $C_i^m + B_i^m$, and then generating new values until it converges on a fixpoint (or fails).

The deadline of message task m_i must be no smaller than the worst-case transmission delay, and we have the following constraint:

$$R_i^m \leqslant d_i^m. \tag{3}$$

Given only the period of each message task, a reasonable assignment of priority would be to give those with shorter periods higher priority. We use this policy for our walk-through example. Returning to our example, the deadlines of the message tasks are determined as follow:

1. We consider a CAN with a bit rate of 100 Kbits/s. The message lengths are assumed to be four bytes for sensor readings and actuator commands and eight bytes for interprocessor communication. From [13], the C^m values are derived as follows: [2]

Message task	Length	C^m (ms)
τ_1^m	4	0.73
τ_2^m	4	0.73
τ_3^m	8	1.3
τ_4^m	8	1.3
τ_5^m	4	0.73
τ_6^m	4	0.73

2. We have five message tasks which have identical transmission periods. Since a CAN requires priorities to be unique, five consecutive priorities are assigned to the message tasks. Message task τ_6^m has a longer execution period than the other five tasks, hence τ_6^m has the lowest priority. Then, the worst-case transmission times are as shown in the table below.

[2] The frame overhead is included in these values.

Task period	Priority	R^m
T_1^m	100	2.03
T_2^m	101	2.76
T_3^m	102	4.06
T_4^m	103	5.36
T_5^m	104	6.09
T_6^m	200	6.82

3. Recall the set of intermediate constraints derived from the phase variable elimination step of Section 4.2.

$$d_3 + d_5 + d_1^m + d_3^m + d_5^m \leqslant 60,$$
$$d_4 + d_5 + d_1^m + d_4^m + d_5^m \leqslant 60,$$
$$d_4 + d_6 + d_1^m + d_4^m + d_6^m \leqslant 80.$$

We select the deadlines of message tasks to be the minimum values that satisfy the above constraints and are no less than the corresponding worst-case transmission times. This allows us to reduce the end-to-end latency. This leads to the following:

$$d_1^m = 3, \quad d_2^m = 3, \quad d_3^m = 5, \quad d_4^m = 6,$$
$$d_5^m = 7, \quad d_6^m = 7.$$

4.4. A heuristic solution review

Fig. 13 pictorially shows the tasks and messages scheduled as a result of our methodology for our walk-through example. The first two time lines denote the periodic sensor readings, while the next two time lines show the periodic control task executions on each host. The periodic updates of

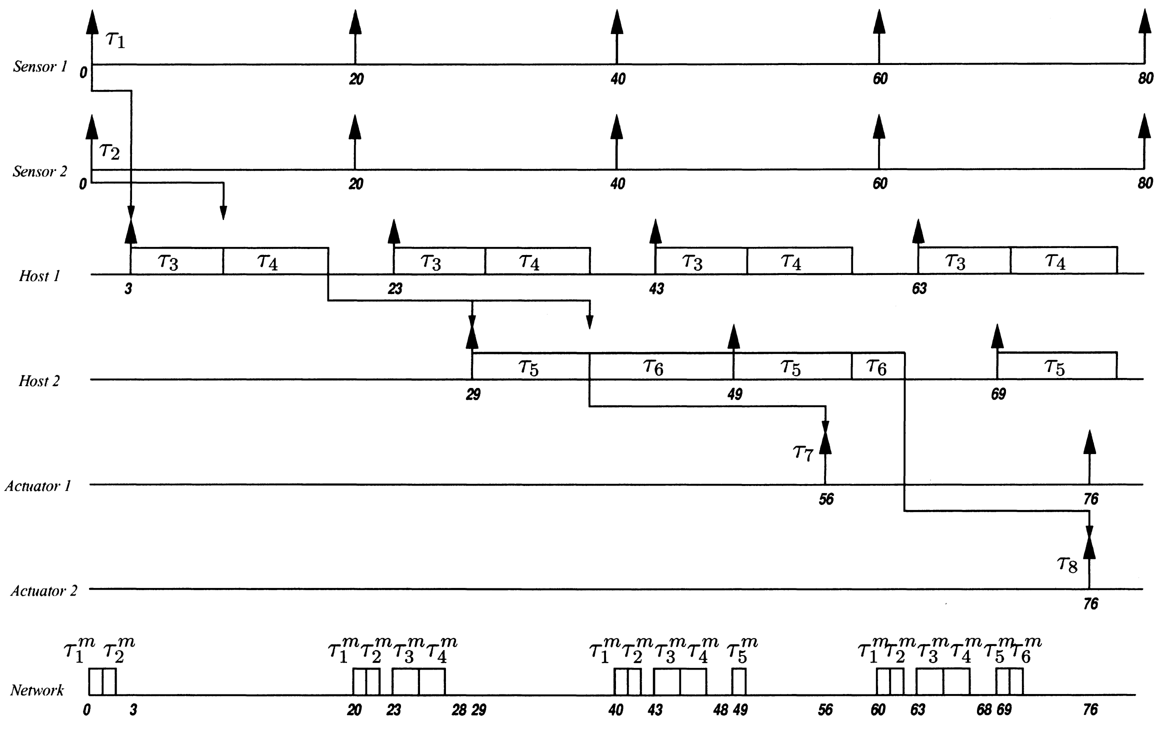


Fig. 13. An example time line: from sensors to actuators.

the two actuators are shown on the next two time lines. Finally, the periodic message traffic on the network is given on the last time line. The scheduling results show that the propagation delay from the sensors to the actuators satisfy the maximum allowable validity time constraints. Furthermore, the other two end-to-end constraints, namely, the input data synchronization constraint and the maximum transaction period constraint, are also shown to be satisfied.

5. Conclusions

We have presented a network conscious approach to designing a distributed real-time system. Given a task graph design of the system, the end-to-end constraints on the inputs and outputs, and the task allocation on a given distributed platform, we proposed a systematic methodology that transforms a high-level design into a system that is schedulable. This was achieved by deriving task specific attributes that eventually meet the end-to-end constraints. This methodology enables developers to streamline the end-to-end design of a distributed real-time system by way of an automatic tool-based approach. We believe that a tool based on our methodology will be helpful for rapid prototyping of designs and identifying design bottlenecks.

There are two directions along which our approach can be extended. First, as a proof-of-concept we would like to implement this methodology as a full-fledged software package. We have already implemented a single processor version and are in the process of extending it into a distributed version. We are looking to apply the methodology in a real-life industrial system. This also has been partially achieved on a single processor system [18].

The other direction we foresee is in extending our design model to incorporate more elaborate task structures, communication mechanisms, and timing constraints. Also, we believe sensitivity analysis is essential. This is because our method works at design time making it difficult to estimate the precise execution times. It is desirable that small changes in execution times do not interfere with constraint solving process.

References

[1] D. del Val, A. Vina, Applying RMA to improve a high-speed, real-time data acquisition system, in: Proceedings of IEEE Real-Time Systems Symposium, IEEE Computer Soc. Press, Silver Spring, MD, December 1994, pp. 159–164.

[2] L. Doyle, J. Elzey, Successful use of rate monotonic theory on a formidable real-time system, in: Proceedings of IEEE Workshop on Real-Time Operating Systems and Software, IEEE Computer Soc. Press, Silver Spring, MD, May 1994, pp. 74–78.

[3] M. Klein, J. Lehoczky, R. Rajkumar, Rate-monotonic analysis for real-time industrial computing, IEEE Computer, January 1994, 24–33.

[4] J. Xu, D. Parnas, Scheduling processes with release times, deadlines, precedence and exclusion relations, IEEE Transactions on Software Engineering 16 (3) (1990) 360–369.

[5] A. Burns, Preemptive priority based scheduling: An appropriate engineering approach, in: S. Son (Ed.), Principles of Real-Time Systems, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[6] K. Ramamritham, J.A. Stankovic, Scheduling algorithms and operating systems support for real-time systems, Proceedings of the IEEE 82 (1) (1994) 55–67.

[7] H. Gomaa, Software Design Methods for Concurrent and Real-Time Systems, Addison–Wesley, Reading, MA, 1993.

[8] B. Selic, Modeling real-time distributed software systems, in: Proceedings of the Workshop on Parallel and Distributed Real-Time Systems, IEEE Computer Soc. Press, Silver Spring, MD, April 1996, pp. 11–18.

[9] L. Sha, S.S. Sathaye, Systematic approach to designing distributed real-time systems, IEEE Computer 26 (9) (1993) 68–78.

[10] R. Gerber, S. Hong, M. Saksena, Guaranteeing real-time requirements with resource-based calibration of periodic processes, IEEE Transactions on Software Engineering 21 (7) (1995) 579–592.

[11] M. Saksena, S. Hong, Resource conscious design of real-time systems: An end-to-end approach, Technical Report ASRI-TR-95-01, Automation and Systems Research Institute, Seoul National University, Korea, November 1995.

[12] T. Rahkonen, Distributed industrial control systems – A critical review regarding openness, Control Engineering Practice 3 (8) (1995) 1155–1162.

[13] K. Tindell, H. Hansson, A. Wellings, Analysing real-time communications: Controller area network (CAN), in: Proceedings of the IEEE Real-Time Systems Symposium, December 1994.

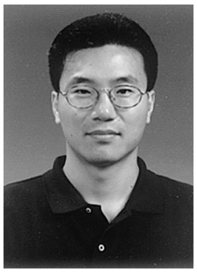
[14] A. Mok, Towards mechanization of real-time system design, in: A.M. van Tilborg, G. Koob (Eds.), Foundations of Real-Time Computing: Formal Specification and Methods, Kluwer Academic, Dordrecht, 1991, pp. 1–37.

[15] D. Stewart, P. Khosla, The Chimera methodology: Designing dynamically reconfigurable and reusable real-time software using port-based objects, Intl. Journal of Software Engineering and Knowledge Engineering, 1996 (to appear).

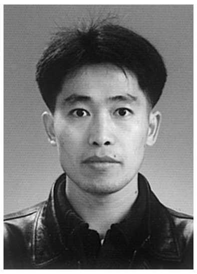
[16] K. Tindell, A. Burns, A. Wellings, Allocating real-time tasks (an NP-hard problem made easy), The Journal of Real-Time Systems 4 (2) (1992) 145–165.

[17] G. Dantzig, B. Eaves, Fourier-Motzkin elimination and its dual, Journal of Combinatorial Theory. Series A 14 (1973) 288–297.

[18] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, H. Shin, Visual assessment of a real-time system design: A case study on a CNC controller, 1996 (under review for publication).

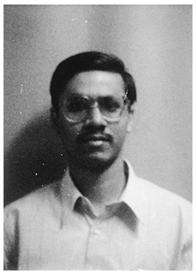


Jung Woo Park received the B.S. and M.S. degrees in Control and Instrumentation Engineering from Seoul National University, Seoul, Korea, in 1989 and 1991, respectively. He is currently a candidate for Ph.D. degree in the School of Electrical Engineering at Seoul National University, Seoul, Korea. His current research interests are real-time systems, industrial networks and performance evaluation of field-bus networks.

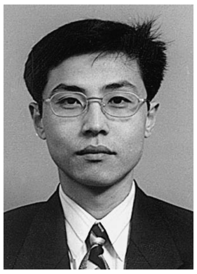


Young Shin Kim was born in Korea on January 15, 1971. He received the B.S. and M.S. degrees in control and instrumentation engineering from Seoul National University, Seoul, Korea, in 1993 and 1995, respectively. He is currently a candidate for Ph.D. degree in the School of Electrical Engineering at Seoul National University, Seoul, Korea. His current research interests are network analysis and application, discrete event system, hybrid system and computer application for factory automation.

Seongsoo Hong is currently a faculty member of School of Electrical Engineering and Automation and Systems Research Institue of Seoul National University in Korea. He received the B.S. and M.S. degrees in computer engineering from Seoul National University, Korea in 1986 and 1988, respectively. Dr. Hong received the Ph.D. in computer science from the University of Maryland, College Park. From April to August of 1995 he worked at Silicon Graphics Inc. as a Member of Technical Staff. From December 1994 to April 1995, he was with the University of Maryland as a Faculty Research Associate. Prior to his work in the USA, he was a Technical Staff Member at Electronics and Telecommunication Research Institute (ETRI), Korea (1988–1989). His current research interests include real-time systems, operating systems, multimedia systems, distributed computer control systems and software engineering. He is now undertaking research projects to develop a systematic methodology for distributed control systems and real-time operating systems.



Manas Saksena is an assistant professor of computer science at Concordia University, Montreal, Canada. He received his Ph.D. from the University of Maryland, College Park. Dr. Saksena has worked in the areas of real-time scheduling, distributed real-time operating systems, and network bandwidth allocation. His current research interests include ene-to-end design and scheduling of real-time systems and system support for media applications.



Sam H. (Hyuk) Noh received the B.S. degree in computer engineering from the Seoul National University, Korea in 1986, and the Ph.D. degree from the University of Maryland at College Park in 1993. He held a visiting faculty position at the George Washington University from 1993 to 1994 before joining Hong-Ik University in Seoul, Korea, where he has been an assistant professor at the Department of Computer Engineering since 1994. His current research interests include parallel and distributed systems, I/O issues in operating systems, and real-time systems. Dr. Noh is a member of the ACM and IEEE Computer Society.

Wook Hyun Kwon (M'76 - SM'88) was born in Korea on January 19, 1943. He received the B.S. and M.S. degrees all in electrical engineering from Seoul National University, Seoul, Korea, in 1966 and 1972, respectively. He received the Ph.D. degree in control from Brown University in 1975. From 1975 to 1976 he was a research associate at Brown University, and from 1976 to 1977 he was an adjunct assistant professor at University of Iowa. Since 1977, he has been with Seoul National University, now as a professor. From Jan. 1981 to Jan. 1982 he was a visiting assistant professor at Stanford University. His research interests are currently multivariable robust and predictive controls, discrete event systems, network analysis, and computer application for factory automation. Dr. Kwon is the director of the Engineering Research Center for Advanced Control and Instrumentation established by the Korean Science and Engineering Foundation.