2

Real-Time Systems

2.1 Background

Nowadays real-time systems have become a common issue in modeling computer systems behavior for time performance. As the approach followed in this book is to present how computer communication affects control law performance, to achieve this strategy, it is necessary to understand how real-time systems can be modeled and measured.

Several strategies comprise real-time systems. These strategies can be classified by two main aspects: the needs and the algorithms of real-time systems. The first aspect allows us to understand why real time is required for some conditions like the presence of the fault and the respective fault tolerance issue. For other conditions like clock synchronization, it is necessary to review real time to achieve a feasible communication performance.

On the other hand, a second aspect is related to how scheduling algorithms are focused into several aspects having an impact on system performance. This is reviewed for consumption time, and it is accomplished by time diagrams.

One of the most important issues for real-time systems is the conformation of time diagrams to define system behavior under several scenarios. This strategy visualizes how the algorithm would perform with certain variations in time. Because this strategy provides the visualization of system response, another issue arises related to how valid is scheduling configuration. This is known as schedulability analysis.

Some other aspects such as load balancing, task precedence, and synchronization are reviewed, which provide an integral overview of modeling real-time systems and what are the repercussions of such an approach.

This revision of real-time systems provides a strong idea of how control law is affected by these time variations, which are the results of several conditions that are beyond the scope of this book. The important outcome of this review is how time delays can be modeled to be defined under the control law strategy.

2.2 Overview

One of the main characteristics of real-time systems is the determinism (Cheng, 2002) in terms of time consumption. This goal is achieved through several algorithms that take into account several characteristics of tasks as well as the computer system in which is going to be executed.

Real-time systems are divided in two main approaches: mono-processor and multiprocessor. These two are defined by different characteristics. The mono-processor has a common resource, the processor, and the multiprocessor has a common resource, the communication link. The last common resource can be challenged through different communication approximations. For instance, the use of shared memory is common in high-performance computing systems in which the use of databuses becomes common in network systems. The multiprocessor approach is the followed in this work.

There are two main sources of information that should be reviewed as introduction to real-time systems: one is by Kopetz (1997) and other is by Krishna *et al.* (1997), both have a review of several basic concepts that are integrated to give a coherent overview of real-time systems, like fault tolerance strategies, the most common protocols, the most common clock synchronization algorithms, as well as some of the most useful performance measures. From this review, one of the most important needs for real time systems is fault tolerance because its performance evaluation is modified to cover an abnormal situation.

Fault tolerance is a key issue that has a lot implications in different fields such as the configuration in communication and the structural strategy to accommodate failures. Most current strategies are based on the redundancy approach, which can be implemented in three different ways:

- Hardware;
- Software;
- Time Redundancy.

Hardware redundancy has a representation known as replication using voting algorithms named N-modular redundancy (NMR). Figure 2.1 shows the basic structure of this type of approach. In this case, several strategies can be pursued.

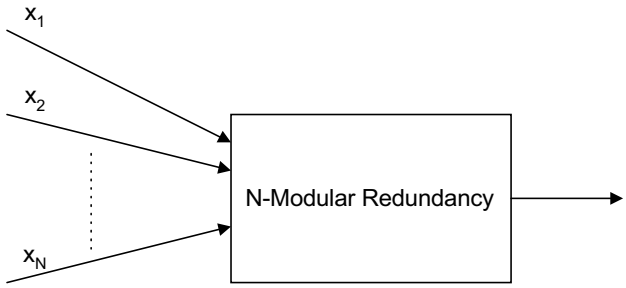

Figure 2.1. N-modular redundancy approach

Different approaches are defined as voting algorithms to mask faults in a trustworthy manner. These are classified into two main groups as safe and reliable algorithms. The first group refers to those algorithms that produce a safe value when there is no consensus between redundant measures. Alternatively, the second group produces a value even in the case of no consensus, so this last approach becomes common when safety is not an issue. Some of the most common voting algorithms are presented next:

- Majority Voter;
- Weight Average Voter;
- Median Voter.

As an example of safe algorithms, the majority voter is presented: This algorithm defines its output as one element of the largest group of inputs with the minimum difference. For instance, consider x_n inputs with a limit ε to evaluate the difference between two inputs $d(x_i, x_j)$. A group g is conformed by those inputs whose difference is lower than the limit ε. This voter can be defined as:

- The difference between two inputs is defined as $d(x_i, x_j) = |x_i - x_j|$;
- Two inputs x_i and x_j belong to g_i if $d(x_i, x_j) < \varepsilon$;
- The largest (in terms of the number of the elements) g_i is the winner, and one element that comprises the group is the output of the voter.

As an example, consider the next group of inputs {1.001, 1.0002, 1.1, 0.99, 0.98, 0.999}, where the selected limit is $\varepsilon = 0.01$. The difference between these elements is presented in Table 2.1.

Table 2.1. Basic table for voting algorithm example

Evaluated Values	**1.001**	**1.0002**	**1.1**	**0.99**	**0.98**	**0.999**
1.001	0	0.0008	0.099	0.011	0.021	0.002
1.0002	0.0008	0	0.0998	0.0102	0.0202	0.0012
1.1	0.099	0.0998	0	0.11	0.12	0.101
0.99	0.011	0.0102	0.11	0	0.01	0.009
0.98	0.021	0.0202	0.12	0.01	0	0.019
0.999	0.002	0.0012	0.101	0.009	0.019	0

From this table there are three groups $g_1 = \{1.001, 1.0002\}$, $g_2 = \{0.99, 0.98, 0.999\}$, and $g_3 = \{1,1\}$, and the output of this voter is any element of g_2 because it is the largest group.

Another safety algorithm is the median voter. This algorithm selects the middle value from the current group of inputs. In this case, the number of inputs has to be odd to select one single input. There are various ways to define this comparison. A common approach is the definition of differences between two input elements $d(x_i, x_j)$, considering x_n inputs, where the difference between two inputs is defined

as $d(x_i, x_j) = |x_i - x_j|$. The maximum difference value is discarded, as are the two related values. This process is kept working until one element is left and declared the output of the voter.

As an example, consider the same group presented in Table 2.1. From this group of elements, there is one drawback because the number of elements is even, which means there is going to be one last pair of values that can be the output of the voter. In this case, any of these values can be selected. The result is shown in Table 2.2 where those values in bold are the winners.

Table 2.2. Results of median voter evaluation based on Table 2.1

Evaluated Values	1.001	**1.0002**	1.1	0.99	0.98	**0.999**
1.001	*0*	*0.0008*	*0.099*	*0.011*	*0.021*	*0.002*
1.0002	*0.0008*	0	*0.0998*	*0.0102*	*0.0202*	**0.0012**
1.1	*0.099*	*0.0998*	*0*	*0.11*	*0.12*	*0.101*
0.99	*0.011*	*0.0102*	*0.11*	*0*	*0.01*	*0.009*
0.98	*0.021*	*0.0202*	*0.12*	*0.01*	*0*	*0.019*
0.999	*0.002*	**0.0012**	*0.101*	*0.009*	*0.019*	0

One example of a reliable algorithm is the weighted average algorithm. This algorithm (Lorczak, 1989) used the inputs x_i from *1* to N to produce an output based on Equation 2.1. In this case, two values are involved, w_i and s. These two values are defined from Equations 2.2 and 2.3.

$$x_o = \sum_{i=1}^{N} \left(\frac{w_i}{s} \right) x_i \tag{2.1}$$

$$s = \sum_{i=1}^{N} w_i \quad \text{for } i, j = 1 \ldots N \text{ and } i \neq j \tag{2.2}$$

$$w_i = \frac{1}{\left(1 + \prod_{\substack{i=1, j=1 \\ i \neq j}}^{N} \left(\frac{d^2 \left(x_i - x_j \right)}{\alpha^2} \right) \right)} \tag{2.3}$$

where $d(x_i, x_j)$ is the difference between two inputs defined as $d(x_i, x_j) = |x_i - x_j|$. α value is a constant degree related to the sensibility of weights involved in Equation 2.1.

From this kind of algorithm one concept of interconnection merges into a fully connected system. This algorithm defines interconnection between all involved components and a group of similar voting algorithms to reduce signal dependency as

shown in Figure 2.2, as well as to mask local faults. However, the price that is paid is an increase in communication time.

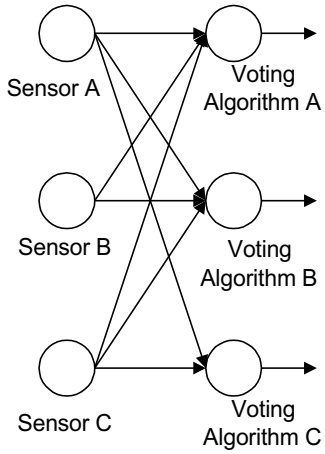

Figure 2.2. Modular redundancy scheme

Alternatively, software redundancy is based on masking software faults, which are different from hardware faults. These are not a consequence of certain conditions during operation. They are the result of design problems in the system. Redundancy becomes an open issue because there is no a proper definition of checking points to evaluate several software versions. In fact, the nature of the faults is defined for design rather than for exogenous effects of time malfunctions.

Certain strategies have been defined like n-version programming (Krishna and Shin, 1997) that are more related to how different programming teams interact to develop software rather than to how algorithms are specifically designed for fault tolerance.

Another common approach is time redundancy, which is defined through recovery points used to roll back system execution when a fault is present. This corrective action takes place when a fault is present, and then the system (or these elements that are affected by the fault) rolls back to a safe point before where the failure occurred. A similar approach is known as rolling forward strategy. In this case, if a fault occurs, those evolved elements go forward up to a safe point where it is known that the system has a fault-free response.

An element that arises as a result of this type of approximation is the evaluation of its performance. This issue alone has become a mature topic. Different approximations have been pursued for fault tolerance and real time. These are defined for reliability, availability, and time performance.

Reliability is defined (Kopetz, 1997) as the probability that a system will provide a certain valid response during a time window.

Availability has been defined as the probability that a system is performing correctly at an instant in time (Johnson, 1989 and Johannessen, 2004). Other performance measures are defined for time consumption and later response.

Another important issue that is similar to fault tolerance is clock synchronization. Several approaches can be pursued, although, some can be eliminated, like current time adjustment, as shown in Figure 2.3.

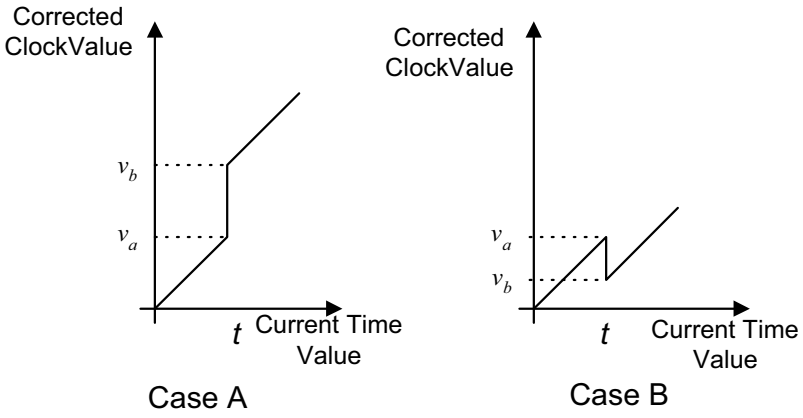

Figure 2.3. Undesirable time correction

From both cases, there is an undesired correction. Case A at time t shows an uncorrect clock value v_a that is corrected instantly to v_b value. This option is not valid due to abrupt forward clock modification and potential loss of current conditions. A similar situation is presented in case B where at time t there is an abrupt backward clock modification that is not acceptable due to loss of current conditions from one point to another.

To avoid this behavior, a common algorithm based on clock skew can be followed using small changes between clocks. This approximation follows Figure 2.4 where correction is performed using skew correction.

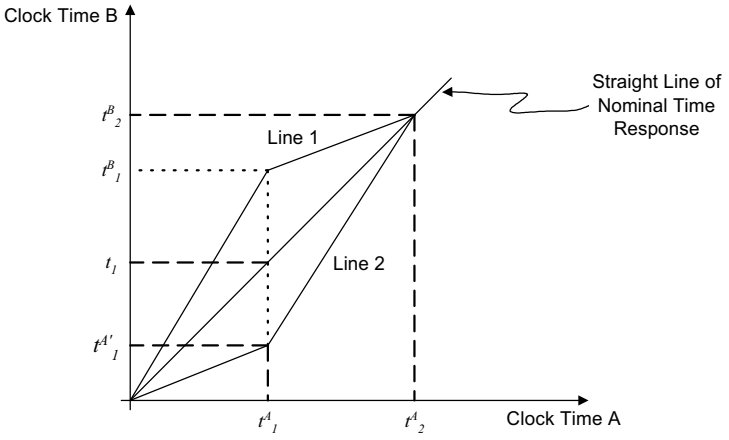

Figure 2.4. Clock skew

In this case, time correction is performed following gradual changes according to a nominal point in time referred to as $(t^B_2,\ t^A_2)$. For instance, consider an observer in Line 2 at t^A_1 where there is a known difference with respect to a nominal time response between t_1 and $t^{A'}_1$. This is corrected by the use of gradual clock modification until t^A_2 is achieved following Equation 2.4.

$$clock\_Time\_B = \left(\frac{t^B_2 - t^{A'}_1}{t^A_2 - t^A_1} \right) * clock\_Time\_A \tag{2.4}$$

In this case, time correction is obtained at t^A_2.

Another algorithm uses a similar principle but in a fault tolerance fashion, in which communication is presented and a comparison between current available clocks takes place in each involved node. This is shown in Figure 2.5. One disadvantage of this approach is related to communication vulnerability, where the time boundary should be present as shown in Figure 2.6. If this boundary is lost by one node, the related clock misses its synchronization and consequently it has to be performed in a broadcast manner. The result of this procedure is a communication overhead caused by time synchronization. However, it presents a reliable response against communication faults.

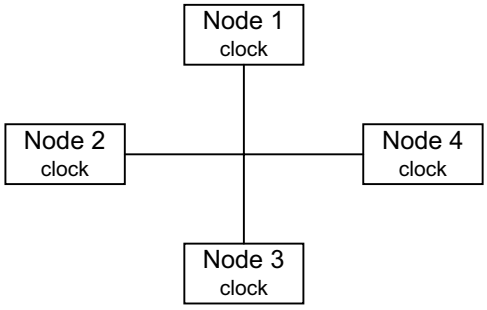

Figure 2.5. Fault tolerance approach for clock synchronization

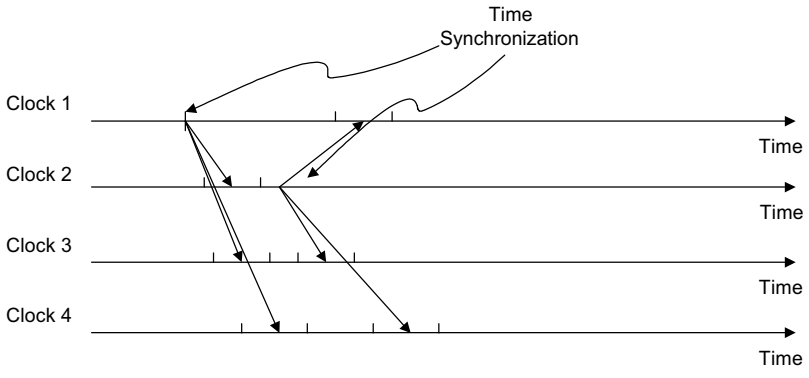

Figure 2.6. Clock synchronization with bounded time

Another strategy is known as the bizantine clock (Krishna, 1997), which is common for intermittent faults (Lönn, 1999). Moreover, real-time systems hold several characteristics that are compatible with other research areas, like discrete control systems. In that respect, real-time control fundamentals have been explored by Törngren (1998), in which the basics are established in common terms such as time delays and time variations in both areas (Table 2.3).

Table 2.3. Common characteristics between computer and control systems

	Discrete Control Systems	Computing Systems
Activation	Time Triggered	Event Triggered
Time Delays	Commonly defined as Constant	Variable
Communication paradigm	Periodic Communication Strategy	Flexible Communication Strategy based upon Scheduling Algorithm
Synchronization	Common Clock Synchronization, Sequential Procedure	Time Stamping Synchronization, Concurrent Programming
Time Variation	Bounded Time Variation	Bounded Time Variation with respect to Scheduling Algorithm.

From these basics, some common time intervals are defined, like communication time, jitter, preprocessing time, and events. Based on these representations, a common graph is defined and referred to as the time graph (Figure 2.7), where the time behavior of those components play a role in communication and need to be represented.

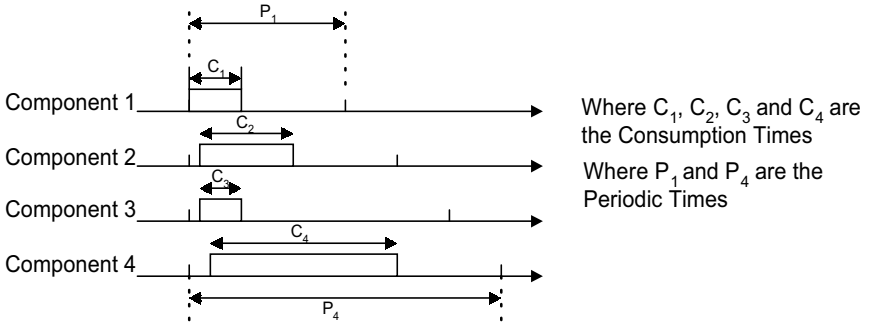

Figure 2.7. Typical time graph

Figure 2.7 presents the classic time graph for four components. This graph shows those time intervals necessary to bound timing behavior from a real-time system. One element that plays an important role in communication is the jitter (*J*) that is

defined as an uncertain time delay, which is a small fraction of any known time delay. It represents the undesirable variation of communication and computing times, in which the case is not clear. Figure 2.8 shows a typical representation of the jitter between elements during communication time.

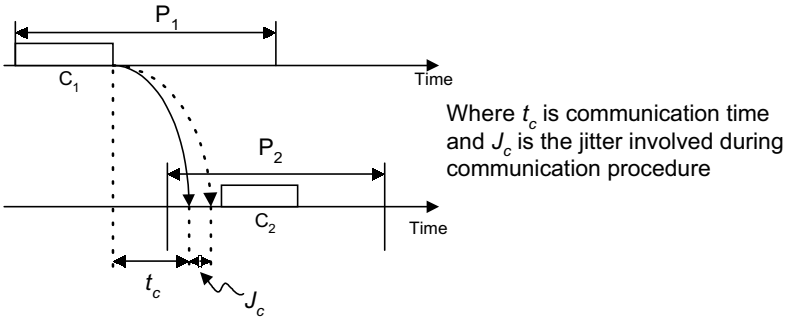

Figure 2.8. Jitter presence during communication performance

From the time graph representation, real time can be measured by adding an action from certain time intervals or events where c_1 is the consumption time of task 1, c_2 is the consumption time of task 2, and p_1 and p_2 are the related periodic times. For instance, take Figure 2.9 as an example, when an event occurs in component 1 and the respective flow chart follows the relation among components 1, 2, and 3, the consumed time (t_{tc}) from this procedure is defined as the sum of all elements involved in a consecutive transmission as shown in Equation 2.5.

$$t_{tc} = t_{c1} + t_{ct1} + t_{pp1} + t_{c2} + t_{ct2} + t_{pp2} + t_{c3} \tag{2.5}$$

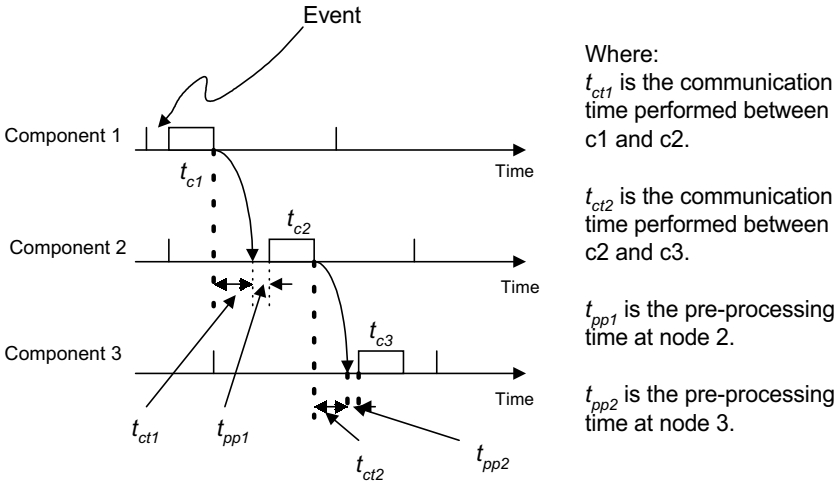

Figure 2.9. Time graph describing communication procedure

In this case, there is no presence of jitter behavior, which in real systems is uncommon. Nevertheless, because communication is bounded through this representation, uncertainties are identified related to jitter assumption. Then, this measure becomes necessary to be known at least through the experience of *ad hoc* equipment knowledge. This graph allows issues like complexity, mutual exclusion, and to a certain extent load balancing, as presented in future sections. An example of this strategy is a fault tolerance approach with clock synchronization as shown in Figure 2.10.

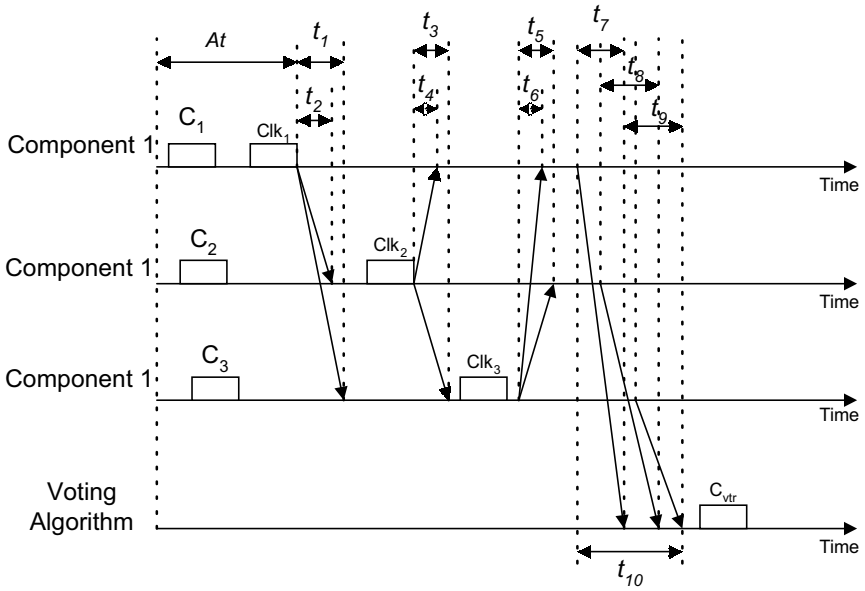

Figure 2.10. Time graph representation of fault tolerance approach

In this case, communication times are t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, and t_9. Consumption times are c_1, c_2, and c_3. Clock measurements are clk_1, clk_2, and clk_3. Consumption time related to the voting algorithm is c_{vtr}. As the reader may realize, these time intervals present an awkward characteristic related to the assumption of maximum communication times like t_1, t_3, t_5, and t_{10} due to a heuristic selection. This maximum final consumption time (T_{CT}) is presented in Equation 2.6.

Having defined every component related to the time graph, these maximum communication times are defined according to protocol and priority definitions.

$$T_{CT} = At + t_1 + Clk_2 + t_3 + Clk_3 + t_5 + t_{10} + C_{vtr} \tag{2.6}$$

This example presents two important cases: one processor sending messages to two different processors and three processors sending messages to one processor.

From this representation, it is possible to define several characteristics such as the need for algorithms that can define time behavior and clock synchronization. In that respect, a class of algorithm called the scheduling algorithm becomes essential.

A real-time system is a multidisciplinary area related to modeling the behavior of a system, to verify its behavior and to analyze its performance. To review various strategies for modeling a real-time system, Liu (2000) and Cheng (2002) have presented a good revision of several scheduling algorithms as well as a formal representation such as deterministic finite state machines.

Moreover, Cheng (2002) presents several formal approximations to verify whether a particular implementation to real-time systems is valid. In this direction, Koppenhoefer and Decotignie (1996) have proposed a formal verification of distributed real-time control based on periodic producer/consumer.

2.3 Scheduling Algorithms

The advantage of using a scheduling algorithm in control systems allows us to bind time delays as well as to define the formal design of their time effects into dynamic systems (Arzen *et al.*, 1999). Moreover, the scheduling strategy sets the boundary of system performance during known scenarios; thereafter, the effects of current control law can be defined off-line without an increase hazardous situations due to inherent timing modification. The incorporation of bounded time delays in control law is reviewed in Chapter 4.

Scheduling algorithms allow us to allocate tasks during a certain time with respect to a common resource such as a processor. These sort of algorithms are defined for the common resource identified, like processors and communication media. The most well-known scheduling algorithms have been defined for the first common resource where there are characteristics to be defined like scheduler analysis. For instance, scheduler analysis for the mono-processor (the processor is the common resource) approach is focused to be less than 1, whereas scheduler analysis for the multiprocessor approach (the communication is the common resource) can be bigger than one depending on the number of nodes that will be involved.

In this section, some of the most typical mono-processor algorithms are reviewed to view task allocation; their counterparts (multiprocessors algorithms) are mainly similar.

Several algorithms can be used, such as rate monotonic (RT), earliest deadline first (EDF), flexible time triggered (FTT) (Almeida *et al.*, 2002), and least slack time (LST). The difference between them is marked by the way tasks are ordered. It depends on the application which method for ordering tasks is the most suitable for a particular example. Those algorithms already mentioned are divided into two categories as static and dynamic schedulers. The main difference is that the static scheduler defines during the off-line process the allocation of task, whereas the dynamic scheduler allocates tasks based on current conditions considering a time slot. For instance, consider three tasks with the next characteristics (Table 2.4 and Figure 2.11). Under the EDF algorithm, if a task changes its deadline at Δt, it would have a higher priority than those tasks already defined (Table 2.5).

Table 2.4. Tasks used to exemplify the EDF algorithm

	Consumption Time (C)	Periodic Time (P)	Deadline (D)	Priority
Task 1 (T_1)	C_1	P_1	D_1	Pr_2
Task 2 (T_2)	C_2	P_2	D_2	Pr_3
Task 3 (T_3)	C_3	P_3	D_3	Pr_1

From Table 2.4, task 3 has the smallest slack time (t_{s3}); therefore, it has the highest priority Pr_1. Thereafter, task 1 has the next highest priority and the last task has the lowest priority Pr_3.

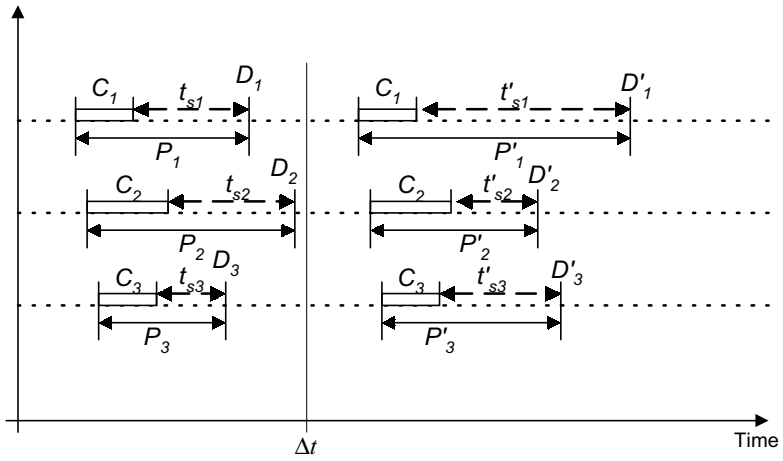

Figure 2.11. Time graph related to Table 2.4

According to Figure 2.11, there are two scenarios for these three tasks. First, task 1 has slack time ts_1, task 2 has slack time ts_2, and task 3 has slack time ts_3, which gives the highest priority to task 3. The second scenario presents a different priority conformation according to slack time modifications.

Table 2.5. New priority order after at reorganization

	Consumption Time (C)	Periodic Time (P)	Deadline (D)	Priority
Task 1 (T_1)	C_1	P'_1	D'_1	Pr_3
Task 2 (T_2)	C_2	P'_2	D'_2	Pr_1
Task 3 (T_3)	C_3	P'_3	D'_3	Pr_2

For the case of deadline modification, as displayed in Figure 2.11 priorities are modified as shown in Table 2.5 where task 2 has the smallest slack time (t_{s2});

therefore; it has the highest priority Pr_1, task 3 has the next highest priority, and the last task has the lowest priority Pr_3.

For real-time purposes, it is best to use static schedulers because of its deterministic behavior. Recently, quasi-dynamic scheduling algorithms have been defined to give certain flexibility to the static communication approach. An example of this sort of algorithm is the planning scheduler (Almeida *et al.*, 1999). The planning scheduler is a pseudo-dynamic scheduler, in the sense that it presents some dynamic properties but is not fully dynamic. The underlying idea is to use the present knowledge about the system (in particular, the variable set) to plan the system activity for a certain time window in the future. Such a time window is fixed, and independent of the periods of the variables, and it is called a plan.

The scheduler must, then, be invoked once in each plan to build a static schedule that will describe the bus allocation for the next plan. The potential benefit of the planning scheduler in terms of run-time overhead is revealed by the following reasoning. Within a fixed time window of duration P_i, such as the period of variable i among a set of N variables, there are at most S transactions

$$S = \sum_{i=1}^{N} \left(\left\lceil \frac{w}{P_i} \right\rceil + 1 \right)$$

When idle time is manipulated to give an opportunity to sporadic tasks, preemptable tasks are to be expected. To perform task re-allocation, the macro-cycle of N tasks is divided into smaller windows called elementary cycles (ECs) that are divided into basic units that are multiples of consumption times of every task. The only condition for an elementary cycle is that it has the same period as the fastest task. As this partition is proposed, the group of tasks conformed by N elements is re-organized according to these time restrictions, taking into account periodic time sizes to define priorities of execution.

If there is one who cannot fit in any EC, it is said that this group of tasks cannot be scheduled.

Some other strategies for scheduling needs can be defined in a more *ad hoc* manner from the basis of the case study; for instance, some scheduling algorithms for control systems have been defined, like Hong *et al.* (2002) and Hong (1995), where the approach is *ad hoc* to the analyzed structure and referred to as the bandwidth-scheduling algorithm. This algorithm proposes a timing analysis of each node time consumption (sensor, controller, and actuator), considering data transmission time and the related time delays. Having established certain time boundaries and the timing analysis of consumption time from every considered element, a review of the proposed algorithm is given. This algorithm consists of ordering elements such as sensors and actuators according to their inherent loop, for instance, sensors, controllers and actuators. Thereafter, this reordering is based on the earliest deadline, first considering the critique and the non-critique zone from each node.

Each scenario has a correspondent control law that considers some time delay conditions like communication time delays from sensor nodes, time consumption from several control nodes, and those considered as sporadic time delays due to non-

real-time messages. Therefore, each modification established by the bandwidth scheduling algorithm has a proper repercussion for the dynamic modeling of the system and, the respective controller. In this case, time delays are bound and used to define the control structure. This is reviewed in Chapter 4. The scheduling algorithm allows us to define the time delay boundary necessary for the control law performance definition.

Alternatively, other *ad hoc* scheduling algorithms have been proposed based on fuzzy logic (Monfared and Steiner, 2000). In this case, a study of the stochastic behavior of the process system is developed. A review of the stochastic nature from different scenarios allows for the use of the adaptive scheduling approach, although it carries the respective uncertainty. This can be tackled by the use of a more restrictive adaptive approach; however, what is recommended by Monfared and Steiner, (2000) is the use of fuzzy logic based on the utilization of several membership functions to represent a Poisson conditional probability function to adapt the best component configuration in terms of the manufacturing control system structure.

Other strategies focusing on hard and soft real-time communication using CANbus have been proposed by Livani *et al.* (1998), where the aim is to divide the message identifier from every CAN word into possible variants called hard real time and soft real time, respectively. Messages are prioritized according to this classification. The accommodation of messages is according to high priority messages (hard real-time messages) as determined by the EDF algorithm. There are four main basic assumptions to be take into account:

- Each real-time message has a reserved time slot.
- The reserved time slot of each message is as long as the worst-case transmission time of the message.
- The priority of a hard real-time message depends on its transmission laxity.

An interesting approach to tradeoff analysis of real-time control including scheduler analysis is proposed by Seto *et al.* (2001), in which a review of optimal control based on the performance index is defined as

$$\left(\max_{u}\right) \quad \left(\min_{u}\right) J(u) = \max \min \left[s\left(x\left(\left(t_f\right), t_f\right)\right) + \int_0^{t_f} L\left(x(t), u(t), t\right) dt \right] \quad (2.7)$$

where $J(u)$ is the performance index.
$S(.)$ and $L(.)$ are the weight functions depending on systems states and control input.
t_f is the final time over the considered interval.
$u(t)$ are the control inputs.
$x(t)$ are the state functions that are dependent on

$$\dot{x} = f\left(x(t), u(t), t\right) \quad (2.8)$$

and having as control input the next function with a related boundary:

$$c(x(t),u(t)) \leq 0 \tag{2.9}$$

This function is minimized based on the system dynamics and schedulability performance. As a result of this optimization, determination of the optimal frequencies for task schedules is performed by solving this nonlinear constrained optimization problem.

For instance, Gill *et al.* (2001) have proposed an approximation for scheduling service based on real-time CORBA middleware. This middleware strategy allows clients to invoke operation without concern for OS/hardware platform, types of communication protocols, types of language implementations, networks, and buses (Vinoski, 1997). Specifically, the strategy pursued by Gill *et al.*, is a scheduling service that has already implemented the most common scheduling algorithms from the static and the dynamic. For instance, this service based on a defined framework has already implemented RM, EDF, maximum urgency first (MUF), and maximum laxity first (MLF), where an abstract implementation is followed based on three main goals:

- Tasks dispatchments are organized by a critical operation that is organized by a static priority in which noncritical operations are dispatched by dynamic scheduling;
- Any scheduling strategy must guarantee scheduling of critical operations;
- The adaptive scheduling approach allows for flexibility to adapt varying application requirements and platform requirements.

It also defines systems requirements by following several steps, which are defined as:

- Any application that gives information used by the TAO scheduling service (implemented as object) to define an IDL interface;
- Time configuration is performed either off-line or on-line as the application demands;
- Scheduling service assigns static and dynamic priority;
- Priorities assigned to each task and the respective subdivision allow for dispatching priority;
- Schedulability is evaluated based on priority assignment and the selected scheduling algorithm;
- Several queues that are necessary to dispatch the already ordered priorities (per node) are created per node;
- Dispatching modules define the thread priorities assignment according to previous analysis.

Following this idea of real time using the middleware structure as resource manager, Brandt and Nutt, (2002) have implemented flexible real-time processing by developing a dynamic quality of service manager (DQM) as a mechanism to operate on the collective quality of service level specifications. It analyzes the collective optimization of processes to determine its allocation strategy. Once the allocation is determined, it defines the level that each application should operate to optimize global performance based on the soft real-time strategy, which an eventual mis-

deadline is feasible. This approximation to real time using middleware presents a competitive task allocation approach, although dynamic management resources still comprise the hard real-time behavior for safety-critical purposes.

Another approach related to real-time middleware is presented by Sanz *et al.* (2001), where an integrated strategy for several functional components like operational control is proposed, such as complex loops, sensors and actuators, monitoring, planning, and execution. This integration is proposed from the perspective of cooperative functional components encapsulated through agents using real-time CORBA. Furthermore, this author has explored the use of "intelligent" strategies for planning the behavior of a complex network control system as presented in Sanz and Zalewski (2003), where design patterns are reviewed to define the most suitable strategies for control law design, task allocation, and exploit design knowledge.

Furthermore, the use of middleware strategies to define suitable scheduling algorithms has been explored by the use of CORBA (Sanz *et al.*, 2001). For instance, the RM algorithm assigns priorities to tasks based on their periods: the shorter the period the higher the priority. The rate of a task is the inverse of its period.

The rate monotonic algorithm behaves as in the following example in the presence of three tasks.

Example: Suppose we have three tasks with the characteristics described in Table 2.6.

Table 2.6. Three tasks for rate monotonic example

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	A	a^T
T_2	B	b^T
T_3	C	c^T

Here T_1 has the smallest period a^T and the smallest consumption time a. The related ordering of this table is presented in Figure 2.12.

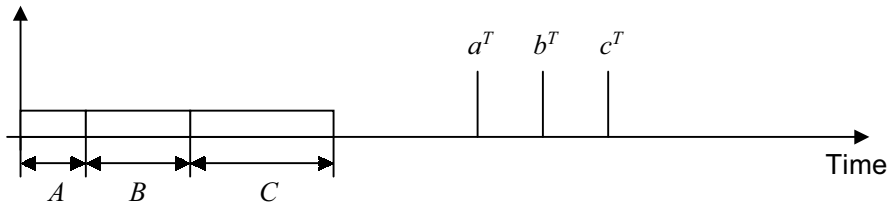

Figure 2.12. Related order from rate monotonic algorithm

Alternatively, we have a case with periodic tasks with submultiples of bigger periods as shown in Table 2.7, where $2a^T = b^T$.

Table 2.7. Another set of tasks for rate monotonic example

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	A	a^T
T_2	B	b^T
T_3	C	c^T
T_4	D	d^T

This group of tasks is organized following the basic principle of this scheduling algorithm with the next distribution (Figure 2.13).

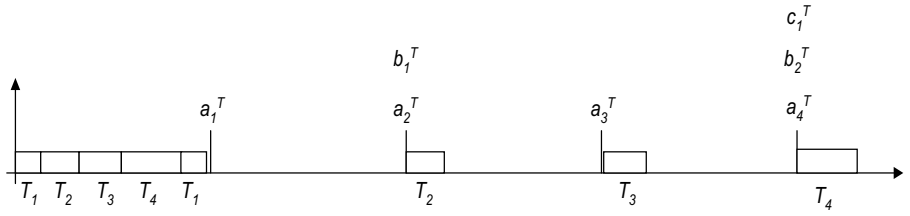

Figure 2.13. Task distribution from Table 2.7 according to rate monotonic

There are two main issues in this algorithm. First, it has a common resource, processor performance, and second, its organization allows for a priority analysis in terms of the capacity to allocate every task following its respective time restriction. This analysis is referred to as schedulability analysis (Liu and Layland, 1973), where the basic condition is that the total percentage consumed by the consumed time (c_i) from every task with respect to its period (T_i) should be less than or equal to one. This condition is expressed as follows:

$$U = \sum_{i=1}^{N} \frac{c_i}{T_i} \leq 1 \tag{2.10}$$

where N is the total number of tasks and U is the total percentage of consumption time. If this condition holds true, it is possible to reorganize this group of tasks as stated before.

There are various conditions to be reviewed for this algorithm. For instance, there is time variation with respect to time deadlines and consumption times in which tasks can be derived into this condition. Devillers and Goossens, (2000) present a review of these variations in which the feasibility problem based on the utilization factor may be possible.

On the other hand, The EDF algorithm assigns priorities to individual jobs in the tasks according to their absolute deadlines. The EDF algorithm performs organization based on the proximity of the deadline with respect to the current consumption time left from each task. It holds the scheduling analysis as the rate monotonic algorithm. As an example, Table 2.8 is presented.

Table 2.8. Task distribution for EDF algorithm

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	A	a^T
T_2	B	b^T
T_3	C	c^T

In Figure 2.14, the current time evaluation is denoted as c_t which is the time when it is decided which task is going to be executed following EDF considerations.

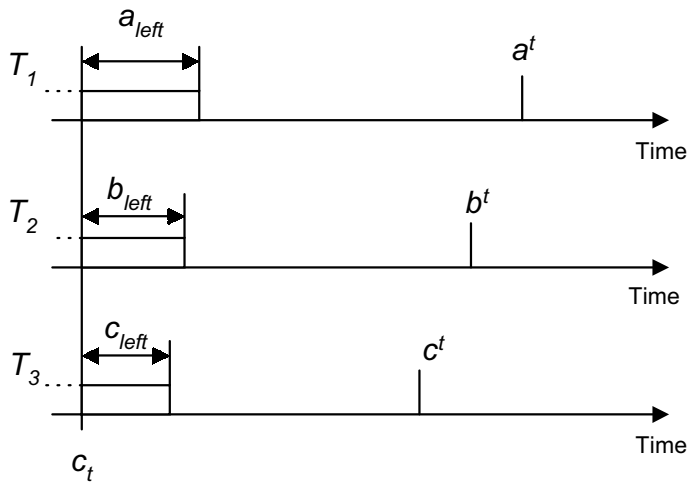

Figure 2.14. Task distribution for EDF example

At c_t, the time left for task T_1 with respect to its deadline is calculated as follows:

$$a^d = a^T - a_{left} \tag{2.11}$$

This procedure is performed for the rest of the tasks:

$$\begin{aligned} b^d &= b^T - b_{left} \\ c^d &= c^T - c_{left} \end{aligned} \tag{2.12}$$

Having produced a^d, b^d, and c^d, a comparison is performed:

$$\begin{aligned} a^b &< b^d \\ b^d &< c^d \end{aligned} \tag{2.13}$$

The smallest value from this comparison is the winner; therefore, it has the capacity to use the common resource (Figure 2.15) until Equation 2.13 is modified.

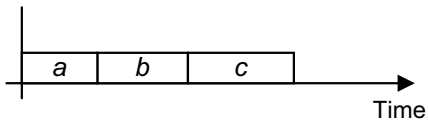

Figure 2.15. Task assignment to common resource according to EDF

The rest of the tasks are organized following the same criteria of deadline evaluation as shown in Figure 2.15. If any task modifies either its consumption time or its deadline condition (Equation 2.13) tasks rearrange priorities and task execution is modified according to new conditions.

The alternative approach for the dynamic scheduling strategy is the LST algorithm. In the LST algorithm, at any time t, the slack of a job (t_s) with deadline at d is equal to d-t minus the time required to complete the remaining portion of the job (Δt) as shown in Equation 2.14 and Figure 2.16.

$$t_s = d - t - \Delta t \tag{2.14}$$

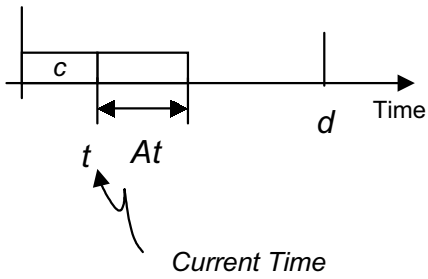

Figure 2.16. Current execution task according to LST

As an example of this dynamic algorithm, Table 2.9 is proposed.

Table 2.9. Task distribution for related example

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	a	a^T
T_2	b	b^T
T_3	c	c^T

Based on Figure 2.17, the monitored system is performed by current time t as follows:

$$\begin{aligned} t_{sa} &= a^T - \Delta t_a - t \\ t_{sb} &= b^T - \Delta t_b - t \\ t_{sc} &= c^T - \Delta t_c - t \end{aligned} \tag{2.15}$$

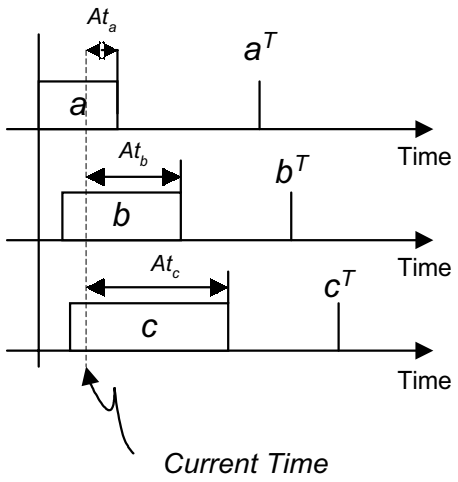

Figure 2.17. Time evaluation according to LST

where t_{sa}, t_{sb}, and t_{sc} are the respective slack time of each task priority; in this example, the task that becomes the common resource is a because t_{sa} is the smallest value and it has the biggest priority. This algorithm has the particularity that it behaves like the EDF algorithm according to certain conditions.

Another dynamic scheduling algorithm is the MUF algorithm. This algorithm organizes a group of tasks following next procedure; it follows the EDF procedure by combining a heuristic priority designation of tasks when both techniques agreed to a certain priority assignment and then the selected task is performed.

It takes into account deadline proximity like the EDF algorithm, and the priority assignment is based on exogenous demands from the case study. As example of this algorithm in Table 2.10 is proposed.

Table 2.10. Task distribution for MUF algorithm priorities

Name of Tasks	Consumption Time (C)	Period Time (T)
T_1	a	a^T
T_2	B	b^T
T_3	C	c^T

Task organization is presented in Figure 2.18.

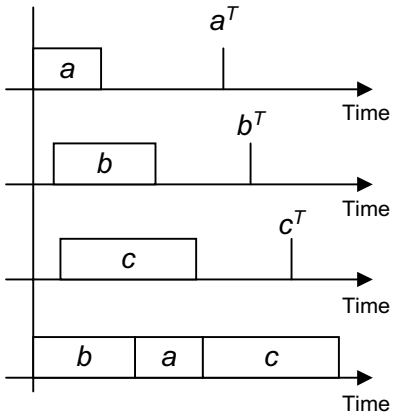

Figure 2.18. Task original organization from Table 2.10

If we check out the result of this reordering (last part of Figure 2.18), it has been performed with an obvious strategy because exogenous priority has won according to Table 2.10.

As a concluding remark from this section, different scheduling algorithms have been presented as well other strategies such as *ad hoc* scheduling algorithm responses and task organization, which provided an overview of the key advantages and disadvantages of different algorithms. One procedure that can be followed to define a feasible approach is presented in Section 2.4. This review gives an idea of how the scheduling approach can be followed taking into account performance or middleware strategies.

Basically, the protocol makes use of the dual-phase elementary cycle concept to combine time- and event-triggered communication with temporal isolation. Moreover, the time-triggered traffic is scheduled on-line and centrally in a particular node called master. This feature facilitates the on-line admission control of dynamic requests for periodic communication because the respective requirements are held centrally in just one local table.

2.4 Distributed Real-Time Systems

Various factors should be taken into account to define a scheduling algorithm for distributed systems. Issues like synchronization arise as fundamentals; for this reason other characteristics are listed next:

- Synchronization;
- Communication cost;
- Load balancing;
- Task assignment and task precedence.

Time synchronization was reviewed in Section 2.3, which defined the most common algorithms like time stamping, passing time synchronization, and sliding linear

regression as determined by Johannessen (2004). Techniques like passing time synchronization have as a main characteristic the use of an inherent protocol with time managing, such as the network time protocol. Cervin *et al.* (2003) have studied the issue of synchronization where synchronization clocks by subnet organizations are commonly recommended although they have a high timing cost that affect performance by inherent time delays.

From this group of possible strategies arises the issue of performance evaluation to determine a suitable approach for certain cases of study; for instance, Lönn (1999) presents a review of different performance evaluation techniques as well as some results with respect to a specific configuration such as fault tolerance average. This approximation presents the best results in average skew and mean time between faults.

This strategy (Kopetz and Oschenreiter, 1987) is classified as a converge clock; it consists of the average of all n-clocks except the *n* fastest clock and the *m* slowest clock. Maximum skew is presented in Equation 2.16.

$$\partial_{\max} = (\varepsilon + 2\rho R)\frac{n-2m}{n-3m} \tag{2.16}$$

where ε is the reading error of a remote clock, R is the resynchronization interval, ρ is the maximum drift between two clocks, and n and m are ∂ from the result of this equation. Clocks are corrected in terms of the ∂_{max} error.

On the other hand, communication cost is defined as the rate between the size of the data to be transmitted and the frequency of transmission. This cost can be defined as a percentage between these two values, although it is based on the characteristics of the case study. For instance, some distributed systems can be loosely connected; therefore they transmit with a very low frequency but with high loaded data in terms of information, such as that reviewed by Coulouris *et al.* (1994). Load balancing is performed when there are various processes and their respective processors; therefore, accommodation is performed by several algorithms that take into account performance measures from both processes and processors. The load balancing algorithm is an *ad hoc* approach to the case study, in which the key factor is how to define performance in terms of the analyzed variables.

There are restrictions related to computational activities in which several processes cannot be executed in an arbitrary order but must take into account the precedence relations defined when the design stage took place (Buttazzo, 2004). These relations are described through an a-cyclic graph, where tasks are represented by nodes and precedence relations by arrows (Figure 2.19).

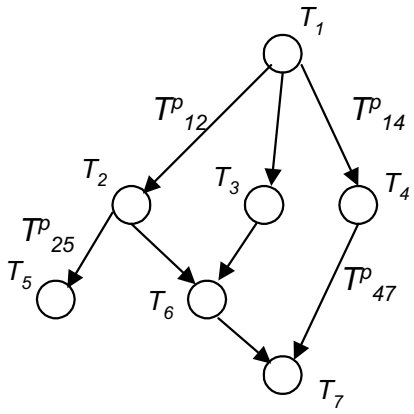

Figure 2.19. Task precedence example

In Figure 2.19, $\{T_1, T_2, ..., T_7\}$ are the nodes and $\{T^p_{14}, T^p_{12}, ..., T^p_{47}\}$ are the related precedence tasks. Task precedence is defined by the case study, and it becomes a requirement for the scheduling algorithm.

There are other implementations like that proposed by Altisen *et al.* (2002), where a compromise between the scheduling algorithm and the control synthesis paradigm is proposed.

As the scope of this work is related to the distributed system, the use of those pieces allow us to cover the main picture of real-time distributed systems, which is to define how time delays can be modeled up to certain conditions like consumption time, which is defined by total time spent from a group of tasks organized by a particular scheduler algorithm. Therefore, an important issue is related to time synchronization between processors; this is performed by various means as shown throughout this chapter. After this review of different algorithms, the proposal of a specific strategy is defined in terms of the needs of the case study in Chapter 5.

2.5 Conclusions

As a concluding remark for this chapter, this was a brief overview of real-time systems and various components such as fault tolerance clock synchronization and scheduling algorithms. We have been reviewed new paradigms in that respect, like the use of middleware in real time and the future directions of this strategy.

http://www.springer.com/978-1-85233-954-8

Reconfigurable Distributed Control
benitez, h.; García-Nocetti, F.
2005, X, 138 p., Hardcover
ISBN: 978-1-85233-954-8