Schedulability Analysis of Distributed Hard Real-Time Systems with Multiple-Event Synchronization

By: J.J. Gutiérrez García, J.C. Palencia Gutiérrez, and M. González Harbour
Departamento de Electrónica y Computadores, Universidad de Cantabria
39005 - Santander, SPAIN
{gutierjj, palencij, mgh}@ctr.unican.es

Abstract[1]

In this paper we present a schedulability analysis technique for distributed hard real-time systems in which responses to different events may synchronize with each other. This technique uses a representation model for distributed systems that allows us to describe not only the task synchronization due to resource sharing, but also the activation due to combinations of events or the generation of several events by a single task. The model is representative of a large number of systems and is suitable for the treatment of message-passing systems or the client-server architecture. The analysis technique is based on the existing RMA techniques for analyzing distributed real-time systems; it allows obtaining upper bounds for the worst-case response times of the system, thus allowing us to make guarantees about the fulfillment of the timing requirements that have been imposed.

1. Introduction

Real-time systems have undergone a notable evolution in the last few years, both in number and variety of applications and in complexity. A result of this evolution is found in distributed real-time systems, which have an increasing importance in today's control systems, since low-cost networking facilities allow the interconnection of multiple devices and their controllers into a single large system.

This paper focuses on real-time systems built using standard operating systems, languages and networks. Most of these systems are scheduled using fixed priorities, and thus the real-time analysis technique that we use is Rate Monotonic Analysis (RMA) [5][4]. Although most of the RMA theory is devoted to single-processor systems, the RMA techniques can also be applied to distributed systems [12][7] by modeling each network as if it were a processor, and each message as if it were a task.

1. This work has been funded by the *Comisión Interministerial de Ciencia y Tecnología* of the Spanish Government under grant TIC99-1043-C03-03

To analyze this type of real-time systems it is necessary to define a model that can represent the system in a precise and, at the same time, simple way. The model should represent, not only the characteristics of the architecture of the distributed system, but also the hard real-time requirements that are imposed. Most of the existing analysis techniques for the scheduling of distributed hard real-time systems are based on a model that we call *linear*, which is representative of a large number of systems. In the linear model each task is activated by the arrival of a single event or message, and each message is sent by a single task. However, this linear model does not allow complex interactions among the responses to different event sequences, except for the shared resource synchronization, and so, the analysis is not applicable to systems in which these interactions exist.

Many real-time analysis techniques take into account complex synchronization and interactions between event sequences, but for scheduling mechanisms different than fixed priorities. For instance [2] is applied in statically scheduled systems, and [6] refers to the dynamic scheduling mechanisms used in the Spring kernel. The real-time model described in [9] is rather similar to the one used in our paper, but focuses mainly on the resource and data usage rather than on the relationships among the different processes or tasks in the system; besides, the schedulability analysis techniques used in that paper are not focused on priority scheduling.

This paper addresses the extension of the RMA techniques for fixed priority systems in which tasks synchronize with the arrival of multiple events or messages from the same or other processors in the system. The model that we use allows complex interactions between multiple events. In order to perform the analysis, we transform the system described with this model, into an equivalent system that is described according to the linear model; in this way, we can use the usual schedulability analysis techniques based upon the linear model [12][7], with the appropriate modifications.

The paper is organized as follows. In Section 2, we provide a quick review of the linear model that is currently used for distributed systems. Section 3 describes our model of representation for distributed hard real-time systems that considers the synchronization among the responses to different event sequences. In Section 4, we develop the analysis technique applicable to the new model, which allows us to estimate the worst-case responses to the events, in order to compare them with the timing requirements imposed on the system. Section 5 shows how this analysis technique is applied to an example of a distributed hard real-time system. Finally, in Section 6 we draw our conclusions.

2. Linear system model

The problem of scheduling and analyzing the timing response of concurrent programs executing in the processor resources is quite well defined. As we mentioned before, we shall assume preemptive scheduling based on fixed priorities, and we will thus be able to use RMA for the analysis of the worst-case timing behavior. The scheduling and analysis of the worst-case timing response in the communication resources are also well established. The majority of the standard communication networks do not adapt well to real-time communications, and do not support priority-based scheduling of messages. However, there are some standard communication networks that have been used to carry out hard real-time communications, such as the token-bus [10], FDDI [1], the CAN bus [11], etc. The analysis of the message traffic in these communication resources is carried out using techniques similar to RMA with the processor resources. Thus, we will treat the messages in the communication resources in the same way as the tasks in the processor resources, except for a small blocking term that must be taken into account because the messages are composed of sequences of packets, each of which is indivisible and non preemptive.

2.1. Assumptions

We define a model of an event-driven distributed system with the following characteristics:

- There is a set of external events sequences (generated by external devices, timers, etc.) that activate tasks that are distributed in different processors.
- Tasks may in turn generate internal events which activate other tasks on the same processor, or activate messages that are sent through a communications network. The arrival of a message to its destination represents an internal event that may activate one or more tasks.
- We suppose that all the sequences of external events arriving at the system are known in advance, that is, the rates at which the events with hard real-time requirements arrive and the activities that these events trigger, as well as their deadlines, are known before the system's execution; system requirements make it necessary to guarantee the schedulability of the system at compilation time.
- We suppose that these events are instantaneous and therefore they will have no influence on the calculation of the worst-case response times of the actions which they activate. This does not mean a loss of generality because the overhead effects of the internal events can be easily included in the analysis as extra execution or transmission time.
- We assume that fixed priority scheduling is used, with a FIFO policy for equal priority tasks (i.e., a task may not preempt another task with the same priority).
- In addition, we suppose that the tasks are statically allocated to the processors, and in the same way that the messages are statically allocated to the communication networks. In many hard real-time systems this represents no limitation because the tasks are linked to specific processors by the presence of special hardware devices that are necessary for their execution.

2.2. Structure of the linear model

In the linear model, each external event arriving at the system generates a response in the form of a sequence of actions. Each action is activated by the internal events generated by the previous actions. The actions may be tasks (or a part of a task) executed in a particular processor, or messages sent through a communications network. Normally, the first action of a response is a task. We will call e_i the external event that activates action a_i, and e_{jk} the internal event that activates action a_k and was generated by action a_j. In the linear model of the distributed system, the actions can only be activated from a single event (internal or external), and can only generate one internal event that may in turn activate another single action in the same or in a different resource (processor or network).

There are different kinds of timing requirements that may be imposed on the actions of the response to an external event. We will call *global deadline* of a particular action the maximum interval of time that may elapse between the arrival of the external event and the finalization of that action (as opposed to a *local deadline*, which is measured relative of the arrival of the internal event). The global deadline of the last action in a response will be called the *end-to-end deadline*. We will call D_{ij} the global deadline of action a_j relative to the arrival of the event e_i, and ED_{ik} the end-to-end deadline for action a_k (which should be the last action in the response sequence) relative to the event e_i. The first subscript in the name of a deadline (i) corresponds to the external event. It is not really neces-

sary in the linear model because each action responds to only one external event, but we introduce it now so that the naming scheme is also valid for the extended model of the distributed system that we will see later.

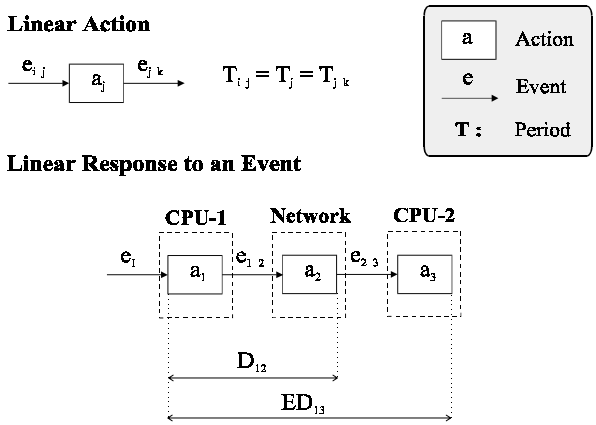


Figure 1. Linear model of an event-driven system

According to this definition of the linear model, Figure 1 shows a diagram of a linear action that is triggered by an input event and that can only generate one output event. The response to the external event sequence is composed of a chain of such linear actions. The period of the external event is inherited by the linear action and also by the output event that it may generate. The figure also shows an example of a response to an external event that is composed of three linear actions (two tasks and a message). Most often, the final action in the response to an external event is a task that is modeled as a special case of a linear action in which the output event is not generated.

3. Extending the distributed system model

In this section we will extend what so far we have called the linear model of distributed hard real-time systems, to a new model that is more general and representative of a greater number of systems. The *multiple-event model* that we are going to define for distributed systems is based on the same principles and assumptions as the linear model, but the new model allows signal & wait synchronization among tasks by permitting their activation through combinations of multiple events, and also the generation of several events by each task. In addition, the multiple-event model takes into account the possible changes in the periods of the actions through rate divisors.

3.1. Structure of the multiple-event model

We will consider the more general case in which an action can generate one or more internal events and thus may activate several actions in the same or in different resources (processors or networks), and which can also be activated by one or several events (internal, external or a combination of both). The timing requirements that we consider are the same as in the linear model, that is, the global deadlines (relative to the arrival of the event) and the end-to-end deadlines (global deadlines corresponding to the last actions in the response to an event). We will continue to call D_{ij} the global deadline of action a_j relative to the arrival of the event e_i, and ED_{ij} the end-to-end deadline (for action a_j relative to the event e_i). It should be noted that in the multiple-event model a specific action a_j can be executed as a consequence of the arrival of different events, and so, it can have several global deadlines corresponding to different events.

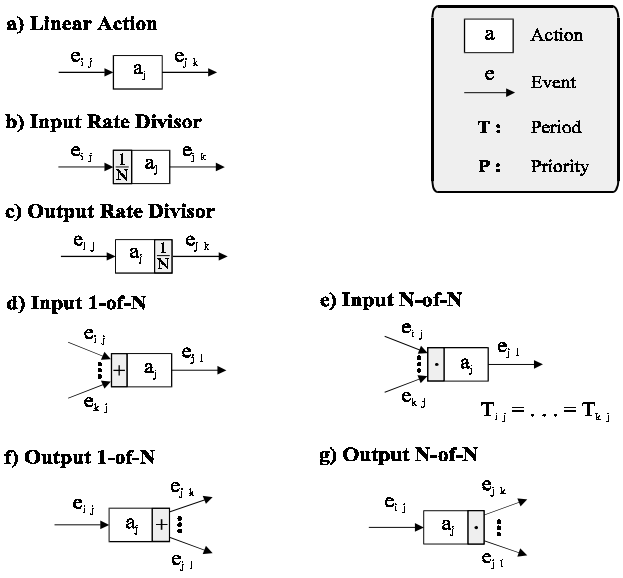


Figure 2. Elements of the multiple-event model

According to the definition of the multiple-event model of distributed systems, Figure 2 shows the different classes of input and output event patterns that we support. Each input event pattern is a combination of events that can activate an action and each output event pattern is a combination of events that are generated upon the finalization of the action. Next, we describe in a more detailed way each of the patterns that we consider in this model:

A) ***Linear Action***. It corresponds to the linear model described in Section 2. The action is triggered by a single event and can only generate one event.

B) ***Input Rate Divisor***. The action is activated when it has received *N* instances of the specified event. An example of this situation is a task that periodically samples a signal, but only processes it when a predetermined number of samples have been recorded.

C) ***Output Rate Divisor***. Only when the associated action has executed *N* times will the corresponding output event or events be generated. An example is the periodic

sampling of a signal, in which each sample needs some preprocessing, and only when a number of samples have been processed is the output event generated.

D) ***Input 1-of-N***. This is the case of an action that can be triggered by any of its input events. As an example for this case we can consider an image processing system in which there are several cameras capturing images periodically. One task per camera preprocesses the images, and then sends the results to a single task which further processes each image. This latter task would be activated each time there was an image available, and therefore it would correspond to the Input *1-of-N* pattern.

E) ***Input N-of-N***. This case corresponds to an action that is triggered by a combination of several events with the same period. The action is only executed when all the input events have arrived. There is no point in considering that the input events have different periods since this would imply the unbounded accumulation of the fastest events. This case corresponds to the situation in which we need to distribute a set of calculations among distinct processors and there is a task that collects the results. The latter task is only executed when all the results are available.

F) ***Output 1-of-N***. This corresponds to the case in which an action can generate one output event out of several possible choices. An example of this case could be a message queue to which a task sends messages (specific data for processing) and there are several tasks available to receive them. The first task that is ready for the reception of a message will be the one that processes the data.

G) ***Output N-of-N***. This is the case in which an action simultaneously generates several events. We can find an example of such an output pattern in a task that distributes a complicated mathematical operation among various processors for parallel execution.

We have not added a rate multiplier because activations of real-time tasks are originated by external events, and thus the execution rates are not faster than the associated rates of the external events.

3.2. Restrictions

Although with the multiple-event model we can describe complex systems using the patterns that we have defined, it is necessary to consider as a restriction the combination by which an event generated by an action with an Output *1-of-N* gives rise to the activation of an action with an Input *N-of-N* at some point of its response sequence. Figure 3 shows this situation and, as we can see, if action a_k or another later action had a global deadline assigned, we could not assure its fulfillment because, since we don't know whether the output event of a_j is $e_{j\ j+1}$ or $e_{j\ j+2}$, we cannot specify the periodicity with which event $e_{k-1\ k}$

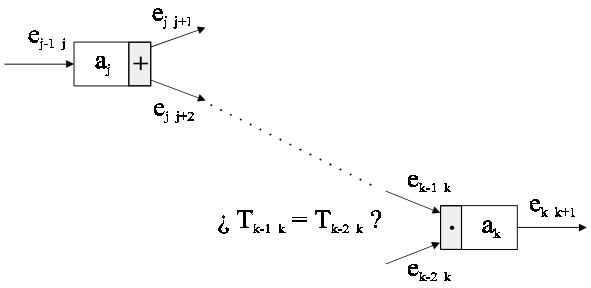


Figure 3. Prohibited event pattern combination

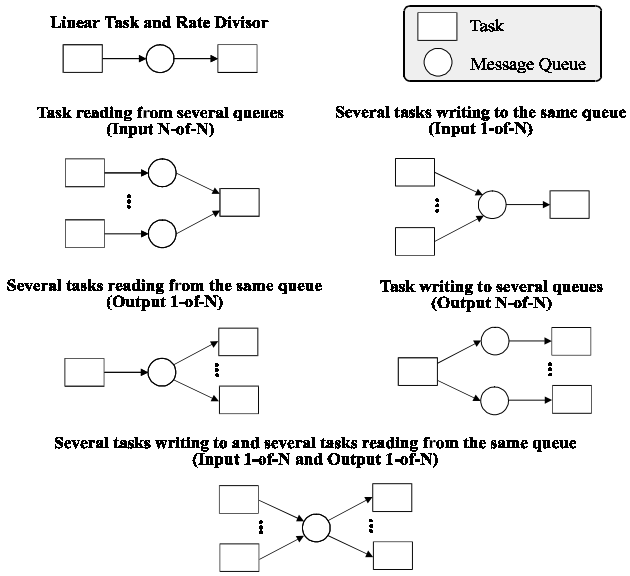


Figure 4. Message queue model

would be generated. We need to consider the condition imposed on the Input *N-of-N* actions, by which all the input events must have the same period. Only if we did not have real-time requirements on the action a_k and later actions then we could allow the existence of this combination of event arrival and generation patterns.

3.3. Applicability of the model

The model described adapts well to the representation of a large number of real-time architectures that can be found in practice. For example, many distributed systems use a message-queue architecture [3]; as we show in Figure 4, using the multiple-event model we can describe many possible situations with this architecture. When two tasks located in the same processor communicate through a message queue, the message will be modeled as an internal event, and a certain amount of execution time will be added to each task to account for the overhead due to the use of message queues.

In the same way as with the linear model, we can consider the example of a system that uses a client-server approach with remote servers. Here as well, we can decom-

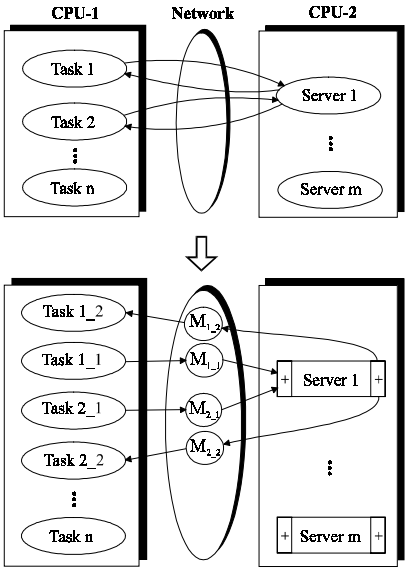


Figure 5. Client-server model

pose each part of a task that requests a service from a remote server in the following sequence of actions: the action before invoking the service, the message sent to the server, the server's action, the reply message, and the action after invoking the server. However, in this case the servers can serve more than one client, and so they will be modeled as a task with an Input *1-of-N* and an Output *1-of-N* with certain restrictions on their output events. These restrictions are introduced to consider the correspondence between inputs and outputs, to avoid the combinations of inputs and outputs that are not possible (each output event corresponds to only one input event, always the same one). Figure 5 shows the model of a distributed system with two nodes and a network, that uses the client-server approach.

4. Real-time analysis of the new distributed system model

To extend the real-time analysis to the multiple-event model of distributed system, we use the analysis techniques that exist for the linear model, which are pessimistic but allow us to guarantee the schedulability of these real-time systems. Thus, the analysis will be carried out applying the RMA techniques applicable to the linear model [7][8][12], extended to take into account the interactions between events. For this purpose, we are going to transform this model into an *equivalent model* for which these analysis techniques can be applied. The equivalent model has a timing response that represents an upper bound for the worst-case response time of the actual real-time system. Hence, below we will see how to extract the equivalent model for each of the event arrival and generation patterns defined in the multiple-event model.

4.1. Equivalent model for the analysis

The equivalent model is a transformation of the multiple-event model of a distributed system that allows the application of the real-time analysis techniques that are used for the linear model. In the transformation to the equivalent model, the parameters (periods and deadlines) corresponding to each of the actions are obtained from the characteristics of the events of the system and of their responses and, furthermore, each input or output pattern is transformed to a new linearized pattern that can be analyzed using the technique developed in [7][8][12] with a few extensions that we will describe here.

Starting from the multiple-event model of a distributed hard real-time system, the equivalent model for the analysis is obtained in the way shown in Figure 6 and Figure 8. Next, we describe in detail the transformations required for each of the event patterns defined in the multiple-event model:

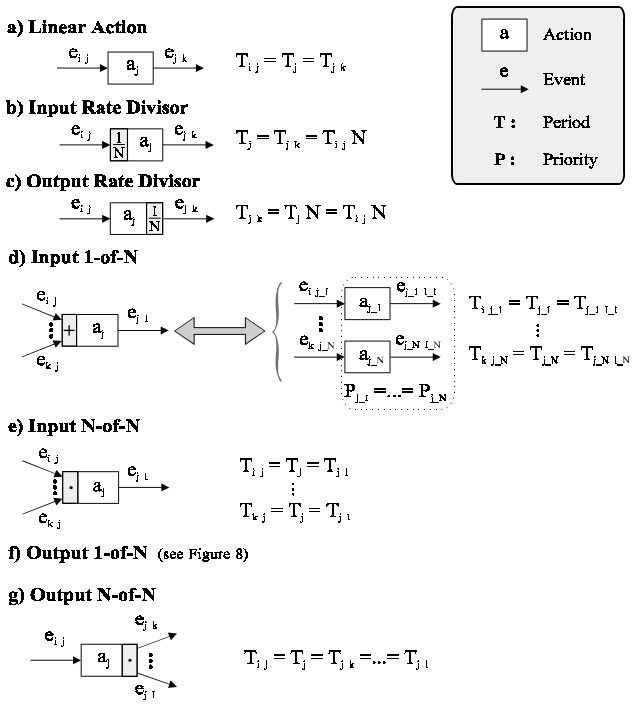


Figure 6. Equivalent model transformations

A) ***Linear Action***. Its own period and that of the event that it generates, are both equal to the period of the input event.

B) ***Input Rate Divisor***. Here the action is activated when it has recorded *N* arrivals of a specific event, and thus, both the periods of the action and of the events that it generates are *N* times the period of the input event. All the actions that are activated after the Rate Divisor have their periods different than the period of the originating external event, and thus it is necessary to define a criterion to establish the

global deadlines. The global deadlines for the actions after the Rate Divisor will be defined according to a worst-case assumption, from the time when the first external event arrives until the execution of the action associated to the global deadline has finalized.

C) ***Output Rate Divisor***. An action with an Output Rate Divisor is activated and executed with the arrival of an event, but only when it has executed *N* times will the corresponding output event or events be generated. Therefore, its period is equal to that of the input event, and the periods of its output events are *N* times larger. The global deadlines are defined in an analogous way to the Input Rate Divisor.

D) ***Input 1-of-N***. In this case the action can be triggered by any of the input events. In the equivalent model we will separate the response sequences associated with each of the different input events in such a way that an analyzable linear structure is obtained. For this purpose, we replicate all the sequence of actions and events starting at the action with the Input *1-of-N* a number of times equal to the number of input events, obtaining independent responses for each of these events (see Figure 6.d).

The replication of an action a_j gives rise to the set of actions $a_{j\_1}\ldots a_{j\_N}$ in the equivalent model, each one belonging to one of the input events. Furthermore, the names $e_{ij}\ldots e_{kj}$ of the input events are changed to $e_{ij\_1}\ldots e_{kj\_N}$, due to the creation of new actions. All the replicated actions are located in the same resource and they are assigned the same priority as the action from which they derive. Each sequence of actions depends on one of the input events and inherits the period of this event. Each action resulting from the replication inherits the global deadline or the end-to-end deadline of the original action for the event to whose response sequence it belongs.

The replication of each chain of actions leads us to an equivalent model with identical response times to that of the original model, if priority scheduling is used with the FIFO policy for equal priority tasks, and the input events are all independent. This scheduling policy would not permit the simultaneous activation of the replicated actions in the equivalent model, since they all have the same priority. In Figure 7 we can see that if the events that activate a_1 and a_2 are queued in FIFO order, the execution sequences in the multiple-event model and in the equivalent model are identical. For example, if $e_{i\ 1}$ arrives before $e_{j\ 1}$, a_1 always attends $e_{i\ 1}$ first and in the same way a_2 will attend the result of $e_{i\ 1}$ first. The equivalent model would execute in the same way, since if $e_{i\ 1\_1}$ arrives before $e_{j\ 1\_1}$, a_{1_1} will be executed before a_{1_2} and in the same way a_{2_1} will be executed before a_{2_2}.

E) ***Input N-of-N***. The case of an action that is executed only when all the input events have arrived (all with identical periods) is modeled in the equivalent model by making the periods of the action itself and all the later actions equal to the period of the input events.

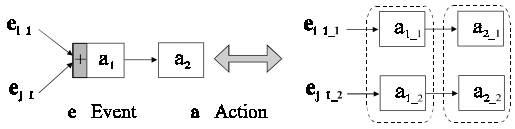


Figure 7. Example of an Input 1-of-N equivalent pattern

F) ***Output 1-of-N***. In the case of an action that can generate different output events, but only one in each period, we will consider three situations in the equivalent model (Figure 8), which give rise to different modes of analysis depending on the periods, deadlines and global worst-case response times of the actions:

F.1) O*utput N-of-N.* We can consider a situation that is always worse than the real system in which instead of generating only one of the output events of the action with Output *1-of-N*, all of them are generated. When calculating the worst-case global responses, actions will be taken into account that will possibly never be executed simultaneously. Therefore, we can pessimistically treat the action with an Output *1-of-N* as an action with an Output *N-of-N*. The periods of the later actions and events are equal to the period of the action with Output *1-of-N*. The analysis in this case will be pessimistic, so next we are going to consider two specials situations (usual in distributed hard real-time applications) in which some of this pessimism can be eliminated.

F.2) *Static Configurations.* For those cases in which all the actions after the Output *1-of-N* in the response sequence to an event have global deadlines less than or equal to the associated periods ($D_{ij} \leq T_j$), we can consider the existence of different configurations of the response, one for each of the events that can be generated by the action with Output

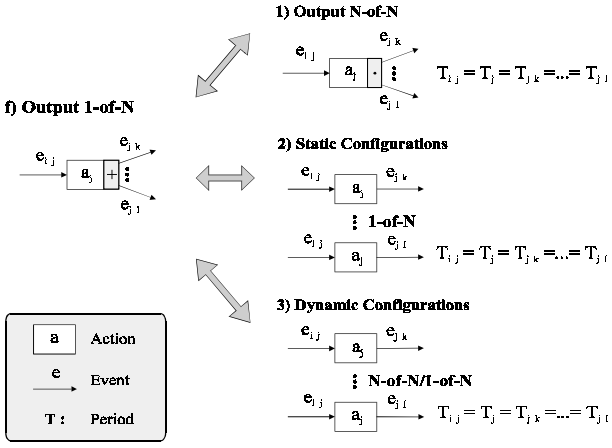


Figure 8. Output 1-of-N pattern in the equivalent model

1-of-N. In each configuration we only consider active those actions triggered by just one of the generated events, and we consider that all the actions triggered by the rest of the generated events are not active. From the point of view of the analysis, it is only necessary to take into account those actions that are active. The periods of the later actions and events are equal to the period of the action with Output *1-of-N*. For the analysis of a given response to an event sequence, each one of its configurations must be analyzed, and only if all the configurations are schedulable we can guarantee the schedulability of the original response.

The restriction of global deadlines less than or equal to the periods guarantees the correctness of the different configurations for the purpose of analysis, since it would be impossible that two response sequences corresponding to two different events generated by the action with Output *1-of-N* were simultaneously active and at the same time, that they met their global deadlines.

The number of configurations in a response to an event sequence depends exponentially on the number of actions with Output *1-of-N* with deadlines within their periods. This means that the configuration analysis can only be carried out for responses with only a few of these actions. For larger responses, the Output *N-of-N* model must be adopted. The configurations that we are considering here are said to be *static* because they can be determined before carrying out the analysis.

F.3) *Dynamic Configurations*. In the previous case we have seen that under special conditions ($D_{ij} \leq T_j$ for the actions of the responses to the output events) we can identify within the model a certain number of static configurations for the analysis. For those actions with deadlines larger than their periods, if we use their global worst-case responses instead of their global deadlines and $R_{ij} \leq T_j$ for them, we could continue to apply the same kind of analysis as in the static configurations, given that it is guaranteed that two response sequences corresponding to two different events generated by the action with Output *1-of-N* cannot be simultaneously active and at the same time meet their global deadlines.

Now however the fulfillment of the conditions that identify the configurations must be carried out during the analysis, since it is necessary to know the global worst-case response times. Thus, starting from an initial situation in which we consider that all the Output *1-of-N* patterns give rise to configurations, we apply an iterative analysis method. For each iteration we identify the valid configurations checking that the associated Output *1-of-N* verifies that $R_{ij} \leq T_j$; otherwise the Output *1-of-N* is changed to an Output *N-of-N*, and a new analysis iteration is applied, until a stable solution is reached. These configurations are called *dynamic*, because they are dynamically identified during the analysis of the system. The number of dynamic configurations cannot be determined in advance.

It is possible that static configurations that are identified before carrying out the analysis coexist in the same system with dynamic configurations that are identified when carrying out the analysis.

G) ***Output N-of-N***. For the case in which an action simultaneously triggers several events, we consider that all the actions and the events following an action with an Output *N-of-N* have the same period as that action. The linear analysis can be applied directly because in the calculation of the global worst-case responses for each of the later actions the rest of the actions are taken into account.

The equivalent model for an Output *1-of-N* operation with restrictions, as in the example of the system which uses a client-server approach shown in Figure 5, can be obtained in two different ways:

- *Starting from the multiple-event model with which we represent the client-server approach*. Considering the restriction imposed on the output events, the replication of actions for this case due to the Input *1-of-N* implies only the replication of the servers and not of the clients. Furthermore, in this replication the connection between the output events and the replicated servers must be done correctly, so that the action invoking the service, and the action executed after the service is completed correspond to the same client.
- *Directly from the client-server model*. Starting from the client-server model, we can arrive at the same equivalent model through the replication of the servers for the clients invoking their services.

4.2. Real-time analysis for the equivalent model

Starting from the multiple-event model of distributed systems, to carry out the analysis we extract the equivalent model in the way described in the previous section. The equivalent model may have replicated actions and events, and may have different static configurations (the dynamic ones will appear during the execution of the analysis itself). For the analysis, we will focus on the following points:

- *Analysis of the configurations*. The configurations appear as a division of a response to an event sequence for the purpose of the analysis, motivated by the actions with Output *1-of-N*. Consequently, in the case where configurations can be identified in a response to an event sequence (static, dynamic, or both), the analysis technique for that response will be based on the application of the analysis for each configuration of the equivalent model. We consider the response to be schedulable when each and every one of its configurations is schedulable.

The response time and jitter analysis of the overall system is performed iteratively, like in the linear system [7][12], with the difference that for each iteration of the analysis, each response configuration and each response without configurations is analyzed by converting the Output *1-of-N* actions of all the other responses into Output *N-of-N* actions. The response times obtained from the analysis are used to obtain the jitter terms of each response, and the analysis is then repeated like in the linear analysis until a stable solution is obtained. This is a pessimistic solution, but ensures obtaining an upper bound for the worst-case response times.

- *Analysis of the Rate Divisors*. The existence of a Rate Divisor in the response to an event must not only be taken into account when obtaining the periods of the actions following it in the event response sequence, but must also be considered in the calculation of the global worst-case responses of these actions. If we consider a Rate Divisor with period T and period factor N, the actions following the divisor in the response sequence will not be executed until the *N-th* activation of the Rate Divisor. Consequently, in the global worst-case response times of these later actions we must take into account an additional term equal to *(N-1)T*, that represents a delay of *N-1* periods. Thus, the impact of the Rate Divisor before the action a_j in the response to the event e_i on the global worst-case response can be expressed as follows:

$$F_{ij} = (N_f - 1)T_{f-1f}$$

where N_f is the period factor of the Rate Divisor closest (starting from the external events) to action a_j, or its own if a_j has an Input Rate Divisor (N_f equals 1 if there is no rate divisor); and $T_{f\text{-}1f}$ is the period of the input event of the Rate Divisor with period factor N_f (if the event were external it would be T_f).

- *Analysis of the Input N-of-N patterns*. These event arrival patterns may give rise to merging the response sequences to different events. These mergers require the incorporation into the analysis of the effects of the associated synchronization, to make the calculation of global worst-case responses valid. Even when all the input events to an action with an Input *N-of-N* have the same period, it is necessary to take into account that the action will be activated on the arrival of the slowest event (the last one to arrive). Supposing that all the external events that give rise to input events to an action with an Input *N-of-N* are generated at the same time, the global worst-case response time for this action would simply be the greatest of the global response times of the action for each external event. However, in real systems we cannot always assure that the generation of the events will be simultaneous, in fact usually they will be out of phase. So, in order to assure the creation of a worst-case situation it is necessary to take into account the maximum phase differences among the external events when calculating the global worst-case response times for the actions with Input *N-of-N*. An identical treatment must be applied to the actions following an action with Input *N-of-N* since, in the same way, they will form part of the response to various external events. We define the phase ϕ_i of each external event e_i as the distance between time zero (an arbitrary instant in which the origin of time is considered), and the instant when the first event is produced. Often the phase is unknown, but it is possible to determine an upper limit for the maximum phase difference among different events, which will be the only factor influencing the analysis. We define the maximum phase difference Φ_{il} between the external events e_i and e_l as the maximum difference in absolute terms between the phases of both events. An upper bound for the maximum phase difference for periodic events that are generated in a continuous way is equal to the period.

  In the analysis of an action a_j with an Input *N-of-N* pattern it is necessary to consider a delay term equal to the maximum phase difference (Φ_{il}) and we must choose the maximum global response time to all the associated external events to whose response the action a_j belongs. We call the set of these external events E_j.

- *Linear analysis*. The cases which we have so far not considered (Linear, Input *1-of-N* and Output *N-of-N*) give rise to a linear-type analysis similar to that used for the linear model. For each external event e_i, we define its period T_i as the inverse of the worst-case rate with which the event can arrive at the system. In the same way, we define the period T_{jk} corresponding to the internal event e_{jk}. The estimation of the global worst-case responses for an action a_j in the response sequence to an external event e_i, as a result of the analysis [7][8] is represented by the term R_{ij}.

Taking into account all the issues that we have mentioned, we will now see how the global worst-case response times are calculated for each configuration or for the whole system in the case when there is only one configuration. Starting from the multiple-event model (and its transformation to the equivalent model) for each action a_j in the response to an external event e_i, we define the global worst-case response time R^*_{ij} (relative to the arrival of the event e_i) in the following way:

$$R^*_{ij} = \max_{l \in E_j} (\Phi_{il} + R_{lj} + F_{ij})$$

where E_j is the set of external events to whose response the action a_j belongs.

Finally, the schedulability of any system configuration (or of the complete system if there are no static or dynamic configurations) is determined through the comparison of the global response times (R^*_{ij}) with the global deadlines (D_{ij}) of the active actions in this configuration.

5. Example: applying schedulability analysis to a distributed hard real-time system

In this section we show an example of a distributed system that responds to the multiple-event model for which the analysis techniques developed in this paper can be applied. Through this example we will show how to obtain the equivalent model starting from the initial specification of the problem, and then how to carry out the analysis. The example is based on a system that was built to detect defects in steel bars using artificial vision, in which our research group was involved. The example has been simplified in order to explain how the analysis is applied.

The problem of defect detection in steel bars is solved by magnetizing the bar, and then submerging it in a solution containing microscopic magnetic particles which, when illuminated with ultraviolet light, emit light in the visible spectrum. The magnetic particles concentrate in greater quantities in the parts of the steel bar that have defects. Thus, the defects of the bar can be seen as different geometric forms that appear on the surface and which can be recognized and analyzed by a vision system. The detection process finalizes with the marking of the recognized defects (with some type of paint for example), and the sorting of the defective and non-defective bars.

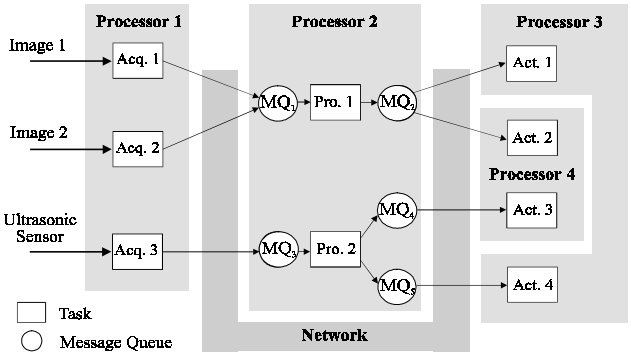


Figure 9. Example of a distributed system

In the artificial vision system of our example (Figure 9) we have four processors that communicate through a single communications network, and which have to carry out the following operations: digitalization of the bar image, defect recognition, and control of the marking and sorting actuators. Our system has two cameras for capturing images and an ultrasonic sensor that carries out an additional inspection of the bar. The two cameras are used to increase the surface of the bar that can be captured each time. The ultrasonic sensor is capable of transmitting an "image" of a segment of the bar in a single block.

The tasks that must be carried out by the different operations in the system are located in the processors according to their associated hardware. *Processor 1* carries out the acquisition of images from the cameras and from the ultrasonic sensor. This processor also does a preprocessing of the images obtained. *Processor 2* is a high-speed processor that detects the defects of the steel bars from the images obtained by the cameras and the ultrasonic sensor. *Processors 3* and *4* have identical characteristics, and control the different marking and sorting actuators of the bars according to the position of the detected defects.

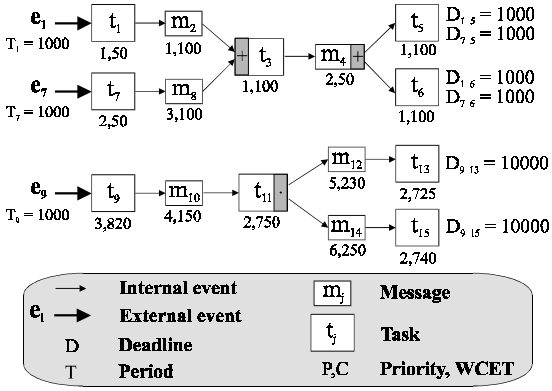


Figure 10. Multiple-event model of the example

As can be seen in Figure 9, the software architecture of the system is composed of a set of tasks that carry out the described operations and which communicate through message queues. All the queues are located in the same processor, and for the analysis we will consider the overhead they introduce to be added to the different actions using them. This system can be represented using the multiple-event model that we defined in Figure 10. The timing requirements are also described in that figure.

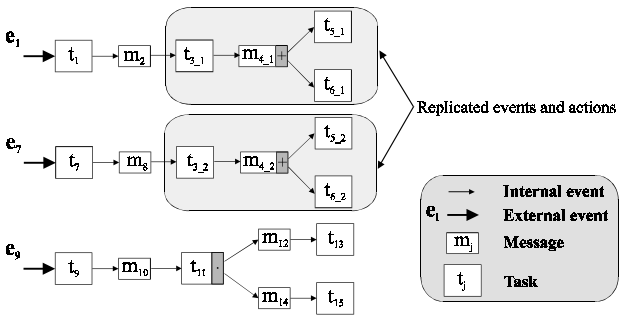


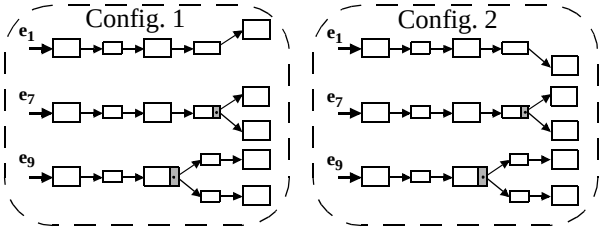
Figure 11. Equivalent model for the example

The analysis begins with the obtention of the equivalent model from the description of the system. First, the actions with an Input *1-of-N* pattern are identified in order to carry

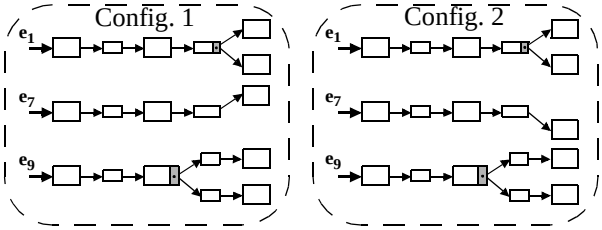
out the replication of actions, according to the procedure described in the definition of the equivalent model (see section 4.1). Figure 11 shows the structure of the response sequences to the external events after the replication of the actions. It can be seen in this figure that action t_3 has an Input *1-of-N* pattern and, therefore, t_3 itself and all the actions following it are replicated.

The following step is the identification of the static configurations resulting from the actions with Output *1-of-N* patterns. Considering the timing characteristics of our system we can identify two static configurations for each of the responses to e_1 and e_7, because each has an Output *1-of-N* pattern and both verify that $D \leq T$ for the last actions in their respective response sequences. Figure 12 shows all the response configurations that we have to include in the analysis of our example.

Analysis of response to e_1:



Analysis of response to e_7:



Analysis of response to e_9:

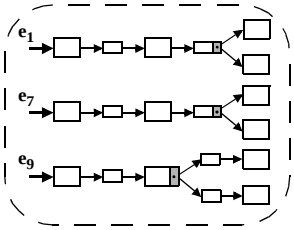


Figure 12. Response configurations in the equivalent model

Now we can perform the analysis of each of the response configurations that we have identified. The worst-case response times obtained as a result of the analysis of each of these configurations for those actions with global deadlines are shown in Table 1. The relevant response times are those in cells with a grey background. As it can be seen, the deadlines are met in every configuration, so all the configurations are schedulable and, thus, the entire system is schedulable. The average utilization of the resources of this distributed hard real-time system is 93.3%.

Table 1. Schedulability analysis results (ms)

Action		t_{5_1}	t_{5_2}	t_{6_1}	t_{6_2}	t_{13}	t_{15}
Deadline		1000	1000	1000	1000	10000	10000
e_1	Cfg1	650	950		850	4575	7090
	Cfg2		850	650	950	4775	6890
e_7	Cfg1	650	950	550		4575	7090
	Cfg2	550		650	950	4775	6890
e_9		650	950	650	950	4775	7090

6. Conclusions

In the work presented in this paper we extend the existing linear model of distributed hard real-time systems to a generalized model that allows the representation of a larger number of distributed real-time systems. This model takes into account the synchronization among tasks, and also their activation by combinations of events and the generation of several events by each task. This model is very suitable for the treatment of message passing systems or for the client-server architecture.

With the definition of the multiple-event model, we have also developed a schedulability analysis technique based on the techniques applied to the linear model. This technique transforms the multiple-event model into an equivalent model over which the linear model analysis can be applied, with the appropriate extensions. Although this analysis technique is slightly pessimistic in some cases, it allows obtaining an upper bound of the system's worst-case response time, thus making it possible to guarantee the fulfillment of the timing requirements of the system.

References

[1] G. Agrawal, B. Chen, W. Zhao, and S. Davari, “Architecture Impact of FDDI Network on Scheduling Hard Real-Time Traffic”. Workshop on Architectural Aspects of Real-Time Systems, December 1991.

[2] G. Fohler, “Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems”. 16th Real-Time Systems Symposium, Pisa, Italy, 1995.

[3] J.J. Gutiérrez García, and M. González Harbour (thesis supervisor), “Planificación, Análisis y Optimización de Sistemas Distribuidos de Tiempo Real Estricto”. PhD Thesis, University of Cantabria, October 1995.

[4] M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour, “A Practitioner's Handbook for Real-Time Systems Analysis”. Kluwer Academic Pub., 1993.

[5] C.L. Liu, and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”. Journal of the ACM, 20 (1), pp 46-61, 1973.

[6] L. Molesky, K. Ramamritham, C. Shen, J. Stankovic, and G. Zlokapa, “Implementing a Predictable Real-Time Multiprocessor Kernel - The Spring Kernel”. IEEE Workshop on Real-Time Operating Systems and Software, 1990.

[7] J.C. Palencia, and M. González Harbour, “Schedulability Analysis for Tasks with Static and Dynamic Offsets”. Proc. of the 19th IEEE Real-Time Systems Symposium, 1998.

[8] J.C. Palencia, and M. González Harbour, “Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems”. Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[9] A.D. Stoyenko, T.J. Marlowe, and P.A. Laplante, “A description Language for Engineering of Complex Real-Time Systems”. Real-Time Systems Journal 11(3) 1996.

[10] J.K. Strosnider, T. Marchok, J.P. Lehoczky, “Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring”. Proceedings of the IEEE Real-Time Systems Symposium, Huntsville, Alabama, USA, pp. 42-52, 1988.

[11] K. Tindell, A. Burns, and A.J. Wellings, “Calculating Controller Area Network (CAN) Message Response Times”. Proceedings of the 1994 IFAC Workshop on Distributed Computer Control Systems (DCCS), Toledo, Spain, 1994.

[12] K. Tindell, and J. Clark, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”. Microprocessing & Microprogramming, Vol. 50, Nos.2-3, pp. 117-134, 1994.