Prioritizing Remote Procedure Calls in Ada Distributed Systems

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

Abstract[1]

In this paper we discuss the assignment of priorities to the execution of remote procedure calls in distributed real-time systems that are programmed using the Distributed Systems Annex (DSA) of Ada 95. We first discuss the current priority model used in the Glade implementation of the DSA. We then present some theoretical results that show that a more flexible priority assignment methodology can provide much better schedulable utilization levels. Based upon these results we propose an implementation that would allow using this priority assignment methodology in Glade. Finally, we propose new features for a future DSA that would allow prioritizing remote procedure calls in a flexible manner.

Keywords: *Ada, Real-Time, Distributed Systems, Priority Optimization, Ada Distributed Systems Annex*

1. Introduction

It is well known that the Distributed Systems Annex (DSA) of Ada 95 [8] provides a flexible way for distributing Ada programs in a multiple-processor platform. This distribution is based on the concepts of program partitions, and remote procedure calls (RPCs). But the language does not provide interfaces nor semantics that would enable using the DSA for distributing applications with real-time requirements. At the time Ada 95 was standardized the mixture of distributed and real-time systems was considered to be insufficiently known, and thus the standard did not attempt its specification. However, the standard is a good starting point for further investigation into these issues. Results from these investigations could perhaps be standardized in a future DSA. This paper explores one aspect of the distribution of real-time applications, which is the prioritization of remote procedure calls.

The implementation of the DSA that is provided with the Gnat free-software Ada compiler is called Glade [9]. In this implementation, an attempt has been made to provide some level of control over the priorities at which remote procedure calls are executed. In section 2 of this paper we review the priority model that we found in the current implementation (version 2.01). As we will show through theoretical results in Section 3, the simple priority model used in Glade would achieve schedulability results that are much poorer than those that can be achieved by using a more flexible priority assignment. Although the problems of priority assignment and of task allocation in distributed systems are NP-Hard problems [6], there exist some heuristic algorithms provide good solutions. One such algorithm for optimizing the priority assignment is called HOPA [1]; it is based on RMA techniques for analysing the response times [3][4][7] and it provides usually good results running in very short times. This is the algorithm that we will use to assign priorities in the distributed system.

In order to be able to apply the desired priorities to the different tasks and remote procedure calls, we need appropriate interfaces between the application and the underlying DSA implementation. In Section 4 of the paper we discuss one way in which we can use existing interfaces to set the desired priorities for RPC's, as well as for the messages sent through the interconnection network, if such priorities are supported.

Finally, Section 5 of the paper proposes changes to the DSA that would enable applications to specify the desired priorities for messages and RPCs in a more convenient and efficient way.

1. This work has been funded by the *Comisión Interministerial de Ciencia y Tecnología* of the Spanish Government under grant TAP97-0892

Of course, the issue of real-time behavior in a DSA implementation depends on many more factors than just the prioritization of the RPC's. These issues will have to be worked out to achieve predictable behavior, but they are out of the scope of this paper.

2. Current Priority Model in the Glade Implementation

The Ada95 DSA requires that "the implementation of the RPC-receiver shall be reentrant, thereby allowing concurrent calls on it from the PCS (Partition Communication Subsystem) to service concurrent remote subprogram calls into the partition." It also requires the implementation to document "whether the RPC-receiver is invoked from concurrent tasks." In addition, it provides the following implementation advice: "Whenever possible, the PCS on the called partition should allow for multiple tasks to call the RPC-receiver with different messages and should allow them to block until the corresponding subprogram body returns."

This basically means that the way in which RPCs are handled is totally dependent on the implementation. In Glade, a pool of tasks is created at initialization time which can take care of concurrently executing the RPC receivers in a given partition. This preallocation of tasks is done for the purpose of avoiding the overhead of task creation and destruction at each RPC. If an RPC arrives and all the tasks in the pool are being used by previously issued RPCs, then a new task will be created for the new RPC. Thus, the Glade implementation follows the advice given in the language Reference Manual.

Since in the DSA there is no provision for expressing the priorities at which the task executing an RPC should execute, the current implementation of Glade provides a simple mechanism that is transparent to the user: the priority of the task invoking the RPC is encoded in the message sent to the receiver partition. Then, in that partition, the task that will execute the RPC reads the priority of the original calling task, which is encoded in the received stream, and sets its own priority to that value before starting the execution of the RPC-receiver procedure. The original priority of the receiver task is set to a medium level. But when the task finishes its work it is left with the priority that it had been assigned for executing the associated RPC-receiver.

One problem with this approach is that the initial priority of the receiver task may cause priority inversion. Either when the receiver task's priority it is set to its initial medium level, or when it is set to a low value because of a previous execution, a high priority request that is about to being serviced by that receiver task may have to wait for a long time, blocked by some other tasks that may in fact have less priority than the request.

Furthermore, as we will see in Section 3, assigning the same priority to all actions in a distributed transaction is a very suboptimal way of assigning priorities.

3. Effects of Priority Optimization in Real-Time Distributed Systems

In this section we will show the influence that the priority assignment methodology has on the overall schedulability of a real-time system. In particular, we will show how the current model used in the Glade implementation, consisting of executing all tasks of a distributed transaction at the same priority level, provides poor results when compared to a method in which these tasks are allowed to have different priorities.

For this purpose, we will show the results obtained on two examples based on a model of an event-driven distributed system. In these systems, there is a set of external events sequences (generated by external devices, timers, etc.) which activate tasks that are distributed in different processors. These tasks may in turn generate events that activate other tasks on the same processor, or make synchronous or asynchronous remote procedure calls which activate messages that are sent through a communications network, further activating the execution of a remote procedure by a remote receiver task. For simplicity, we are going to assume there are only end-to-end deadlines associated to the last task in a response to an event.

For the first example, we will consider a simple distributed system with two processors and a communications network, in which two distributed transactions are executed as a response to two periodic external event sequences. Each transaction contains one task that performs some computation and, as its last instruction, executes an asynchronous remote procedure call (APC). This call implies sending a message, and then executing the call in the remote processor, by a remote receiver task. This call is modelled as a regular task execution. Therefore, the model of the transaction contains two tasks, and one message sent from one to the other. A periodic external event e_i with period T_i activates each transaction, composed of a set of tasks and messages, each with its own worst-case execution time, C_{ij}, and priority, P_{ij}. Furthermore, the last task on the transaction has and end-to-end deadline ED_i. Figure 1 shows the structure of the model that we use for this simple example.

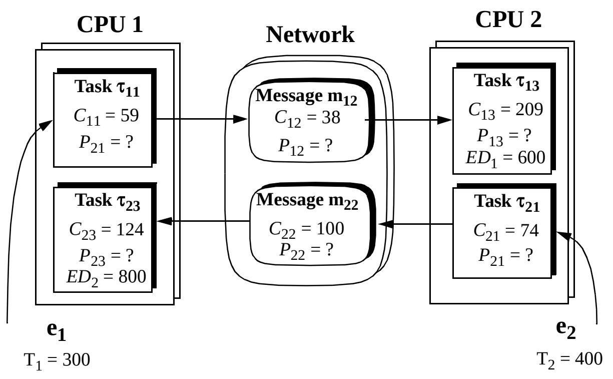

Figure 1. Simple distributed system with real-time constraints

The only parameter that is still to be determined to perform the timing analysis is the priority of each task and message. The possibilities for assigning priorities according to the current Glade implementation are just two: higher priority for Transaction 1, or for Transaction 2. However, if we could assign different priorities to each task and message, the number of different priority combinations would be eight. In order to determine which is the best approach, let us measure the schedulability under the following three priority assignments:

- *Assignment 1*: Rate Monotonic, i.e., in priority order. In the example, if we use the Glade priority approach, the highest priority would be for Transaction 1.
- *Assignment 2*: Higher priority to Transaction 2. This is the second possible assignment in the Glade implementation.
- *Assignment 3*: Higher priority to tasks τ_{11} and τ_{21}, and for message m_{12}. This is the result of applying the HOPA priority optimization algorithm [1] to the example.

Table 1 shows the worst-case response times (WCRT) for

Table 1. Timing analysis

Priority Assignment	WCRT of τ_{13} (ED of τ_{13})	WCRT of τ_{23} (ED of τ_{23})
Assignment 1	306 (600)	813 (800)
Assignment 2	604 (600)	298 (800)
Assignment 3	380 (600)	395 (800)

tasks τ_{13} and τ_{23} when the RMA technique is applied to the three different priority assignments described above. We can see that both of the priority assignments that follow the Glade model make the example unschedulable, but that the third priority assignment allows both transactions to meet their end-to-end deadlines. Thus, the ability to assign different priorities to each component of the system independently, offers better possibilities for achieving schedulability.

With this simple example we have shown the influence of the priority assignment on the schedulability of distributed real-time systems. We will now show that with a flexible priority assignment it is also possible to effectively increase the schedulable utilization of the system.

For this purpose, we will use a simulated example of a system that has to respond to seven external periodic events. Each of these events activates one transaction, each of which has one task which does one or more remote procedure calls. The messages exchanged to perform these remote procedure calls are also considered to be part of the associated transaction. We model each execution of a remote procedure call as a task execution, activated by the arrival of the message generated by the RPC. The total number of equivalent tasks in the model of this system is 50, and the total number of messages is 43. Each transaction has an end-to-end deadline, which we will first assume to be equal to 10 times the respective transaction periods. We pick this number because in distributed systems, to achieve high utilizations it is necessary that the deadlines are larger than the periods by an amount roughly similar to the number of resources used by each transaction.

All tasks and remote procedures are statically allocated in 8 processors, and we will assume that we have a variable number of communication networks, ranging from one to seven. By using different numbers of networks we can check whether there is any influence of the amount of such resources on the system schedulability achieved with the different priority assignments.

The objective of this example is to establish a comparison of the maximum schedulable average utilization that may be achieved with the following priority assignments:

- RM priority assignment, with the same assignment for all tasks and messages in the same transaction. This is the assignment that we would use with the current Glade implementation.
- HOPA priority assignment. In this case, each task or message may have its own independent priority.

In order to determine the maximum schedulable utilization we will start with a system utilization of 1%, and we will use a simulation tool that will increase randomly the worst-case execution times of the tasks and messages, in small steps. At each step we perform the schedulability analysis and the HOPA priority optimization, and we continue until

Table 2. Maximum schedulable utilizations (average)

Number of networks	RM priority assignment(%)	HOPA priority assignment(%)
1	9.175	29.884
2	17.453	35.787
3	21.355	44.840
4	26.660	48.992
5	30.558	55.788
6	32.468	57.515
7	35.105	53.505

we find that the system is no longer schedulable. We repeat this experiment many times for each of the priority assignments and for different numbers of communication networks. Table 2 shows the average results obtained for the maximum schedulable utilization.

These results show that, independently of the number of networks, we can always get much better utilization values by using the HOPA priority assignment. This is in agreement with the fact that with the Glade implementation only 5040 different priority assignments can be done for this example, while with the independent priority approach the number of possible priority assignments ranges between 10^{45} for seven networks and 10^{77} for a single network.

One problem that appears in distributed systems is the effect of deferred activation of tasks or messages, also called jitter. The activation time of messages generated by the execution of periodic tasks is not perfectly periodic, but depends on the completion time of the triggering task, which is variable. The same happens with tasks activated from messages. In general, jitter in one task causes delay in the worst-case response time of lower priority tasks. To solve this problem, there exist scheduling techniques like the sporadic server that can eliminate the negative effects of jitter on the schedulability of lower priority tasks. These techniques can be applied both to the tasks [5] as well as the messages [2].

Table 3. Maximum schedulable utilization in a jitter-free system

Number of networks	RM priority assignment(%)	HOPA priority assignment(%)
1	18.433	100
2	31.687	100
3	42.597	100
4	50.602	100
5	61.428	100
6	67.135	100
7	66.748	100

We have repeated the simulation presented above but assuming a jitter-free system to increase the schedulability. The results appear in Table 3. In this case we can see that the increase in maximum schedulability is larger than when we suffered jitter, as is the total schedulability level that can be achieved.

Similar results are obtained when the ratio of deadlines over transaction periods is lower. Table 4 shows the results for the cases of ED_i/T_i equal to one, three and five, for a system with four networks, both with jitter and jitter-free scheduling.

Table 4. Maximum schedulable utilizations for different end-to-end deadlines

ED_i/T_i	RM (%)	HOPA (%)	RM jitter-free (%)	HOPA jitter-free (%)
1	8.004	19.045	8.204	19.937
2	17.547	34.437	24.882	44.603
3	21.438	43.660	40.270	63.882

4. Proposed Changes to the Glade Implementation

Given the problems mentioned in Section 2 relative to the handling of priorities, we can provide a simple solution by slightly modifying the code of the Glade implementation.

First, to solve the problem of the initial priority of the receiver task, we can just set it to the maximum priority (`System.Priority'Last`). In this way, the receiver task will start at a very high priority, and will immediately read the desired priority from the stream and set its priority to the appropriate value. Later, when the task finishes executing the RPC receiver, it sends back the associated message (in case of a synchronous RPC), and then it will set its priority back to the highest level just before blocking itself waiting for the next RPC request. No priority inversion occurs using this scheme. The only drawback is that this introduces a small overhead in the form of a blocking term for all tasks in the system. But this blocking is small, and does not accumulate to other similar blocking effects: we just have to pick the maximum of all [10].

The other problem of assigning the priorities in a more flexible way, needs some kind of interface that would allow the application to set the priority at which each RPC should be executed in the remote partition. Also, if a priority-based communication system is used, the priorities of the outgoing and the incoming messages should be specifiable.

In order not to add any new interfaces to the PCS, one way of handling the specification of these priorities is to create three task attributes using the facilities described in the optional but standard package Ada.Task_Attributes. These attributes support specifying for each task the priority of the RPC handler task, of the outgoing message, and of the incoming message, for the next RPC. The calling task can change the values of these parameters anytime, for example before each RPC invocation.

For this purpose, we create the following package:

```
package Global_Priorities is

   pragma Pure (Global_Priorities);

   type Priority is range ...

   type RPC_Priorities is record
      RPC_Handler,
      Outgoing_Message,
      Incoming_Message : Priority;
   end record;

end Global_Priorities;
```

In this package, the type Priority represents a value with a global meaning across the distributed system. For each CPU and network, a mapping function exists to translate a value of this global priority type to a value of System.Priority or of the network priority appropriate for that resource. Using the RPC_Priorities type we now instantiate the following package:

```
   package RPC_Attributes is new
      Ada.Task_Attributes(
         Global_Priorities.RPC_Priorities,
         <Initial_Values>);
```

We make this instantiation at the library level, so that it is visible both by the body of the PCS package (System.RPC) and also by the bodies of the application tasks.

The pseudocode of the application task, at the point of an RPC would be:

```
   ...
   RPC_Attributes.Set_Value(
      RPC_Priorities'(
         RPC_Handler       => <value>,
         Outgoing_Message  => <value>,
         Incoming_Message  => <value>));
   Invoke RPC;
   Optionally set the task's priority
      to a new value;
   ...
```

And the implementation of the PCS would be changed in the following way:

- Do_RPC: Instead of writing the priority of the calling task in the stream, we need to read the RPC_Priorities attribute, write the RPC_Handler and Incoming_Message priorities into the stream, and send the message at the Outgoing_Message priority (after mapping it to the appropriate type).
- Do_APC: We do the same, except that the Incoming_Message field is not encoded in the message.
- Body of the task receiver: Before executing the RPC receiver, we read the RPC_Handler (and the Incoming_Message priority if it is a synchronous RPC) and we set its own priority to the mapping of the received value to the appropriate value of the type System.Priority in that system. Later, we can set the priority of the return message.

With the use of the task attributes facility we can pass the desired values for the RPC priorities between the application task and the PCS implementation in a safe way, without adding new interfaces. Since a given application task can only invoke one RPC or one APC at a given time, and since Do_APC and Do_RPC are executed by the calling task, there are no race conditions with the use of these attributes.

5. Proposed Changes for a Future DSA

If a future DSA is to support real-time distributed systems, new optional features and specification needs to be added to the standard. In order to support the proper prioritization of RPCs, according to the results of this paper, a flexible mechanism for specifying the priorities of the RPC's and of the associated messages is needed. The use of task attributes can only be considered as a workaround, and more appropriate interfaces should be specified in the standard.

The problem with RPCs is that they may be invoked many times, from many different tasks with different timing requirements, and thus each invocation may need its own priority. Therefore a priority pragma attached to the specification of the remote procedure is not appropriate.

We need an operation that the user can invoke before making an RPC. This operation should be in a package that is visible from the application. Package System.RPC does not seem appropriate, since it is not intended for direct use from the application. Therefore a new package like the following could be created:

```
with Ada.Global_Priorities;
use Ada.Global_Priorities;

package Ada.RPC_Priorities is

  procedure Set
    (RPC_Handler : in Priority);
```

```
   procedure Set
     (RPC_Handler,
      Outgoing_Message : in Priority);
   procedure Set
     (RPC_Handler,
      Outgoing_Message,
      Incoming_Message : in Priority);
   procedure Get
     (RPC_Handler,
      Outgoing_Message,
      Incoming_Message : out Priority);
end Ada.RPC_Priorities;
```

Procedure `Set` in the above package would set the priority or priorities used for future RPCs or APCs issued by the calling process. These priorities would be in effect until `Set` is called again. In this way, the application can specify the priorities of its RPCs either on an individual basis, or by grouping several calls under the same priorities. Initial values for the priorities should be intermediate values of the respective priority types. Procedure `Get` would return the current values of the RPC priorities.

It could be argued that requiring the user to set the priorities before making an RPC violates the intention of the DSA in the sense that we would be making the procedure call explicitly remote. However, in real-time applications it is absolutely necessary to know the architecture of the partitions and the RPCs, because the fact that a procedure call is or is not remote has a very significant impact on the timing response. One could always use a high level configuration tool that would create the partitions, do the real-time analysis, and insert the appropriate calls to `Ada.RPC_Priorities.Set` before each RPC.

6. Conclusions

In this paper we have proposed a prioritization scheme for remote procedure calls in distributed Ada real-time systems. We have shown through theoretical results how this prioritization scheme works much better than the one provided by the current Glade implementation. We have also discussed how to modify Glade to accommodate the new priority scheme. Finally, we propose an amendment to the Ada 95 DSA, that would allow a flexible assignment of priorities.

It is worth mentioning that RPC prioritization is just one issue among many that have to be worked out to achieve a distributed systems implementation that offers a fully real-time behaviour.

References

[1] J.J. Gutiérrez García, and M. González Harbour: “Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems”. Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, CA, pp. 124-132, April 1995.

[2] J.J. Gutiérrez García, and M. González Harbour: “Minimizing the Effects of Jitter in Distributed Hard Real-Time Systems”. Journal of Systems Architecture 42, pp. 431-447, October 1996.

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

[4] J.C. Palencia Gutiérrez, J.J. Gutiérrez García, and M. González Harbour: “On the Schedulability Analysis of Distributed Hard Real-Time Systems”. Proceedings of the 9th Euromicro Workshop on Real-Time Systems, Toledo, pp. 136-143, June 1997.

[5] B. Sprunt, L. Sha, and J.P. Lehoczky: “Aperiodic Task Scheduling for Hard Real-Time Systems”. The Journal of Real-Time Systems, Vol. 1, pp. 27-60, 1989.

[6] K. Tindell, A. Burns, and A.J. Wellings: “Allocating Real-Time Tasks. An NP-Hard Problem Made Easy”. Real-Time Systems Journal, Vol. 4, No. 2, pp. 145-166, May 1992.

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

[8] S. Tucker Taft, and R.A. Duff (Eds.) “*Ada 95 Reference Manual. Language and Standard Libraries*”. International Standard ISO/IEC 8652:1995(E), in Lecture Notes on Computer Science, Vol. 1246, Springer, 1997.

[9] L. Pautet and S. Tardieu, “Inside the Distributed Systems Annex”, Intl. Conf. on Reliable Software Technologies, Ada-Europe’98, Uppsala, Sweden, in LNCS 1411, Springer, pp. 65-77, June 1998.

[10]J.B. Goodenough, and L. Sha. “The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Tasks”. Proceedings of the 2nd International Workshop on Real-Time Ada Issues, June 1988.