Real-Time Distribution Middleware from the Ada Perspective[1]

Héctor Pérez, J. Javier Gutiérrez, Daniel Sangorrín, and Michael González Harbour

Computers and Real-Time Group
Universidad de Cantabria, 39005 - Santander, SPAIN
{perezh, gutierjj, daniel.sangorrin, mgh}@unican.es
http://www.ctr.unican.es/

Abstract. Standards for distribution middleware sometimes impose restrictions and often allow the implementations to decide on aspects that are fundamental to the correct and efficient behaviour of the applications using them, especially when these applications have real-time requirements. This work presents a study of two standard approaches for distribution middleware that can be used from Ada applications: RT-CORBA, and the Distributed Systems Annex (DSA) of Ada. The study focuses on the problems associated with the real-time behaviour of some implementations of these approaches, and on possible solutions that can be derived from our experience with Ada implementations. Moreover, the paper considers the problem of integration of the distribution middleware with a new generation of scheduling mechanisms based on contracts.

Key words: distribution middleware, real-time, communications, RT-CORBA, Ada DSA, performance.

1 Introduction

The concept of a distributed application is not new; it has existed since two computers were first connected. However, the programming techniques of these systems have evolved greatly and they have become especially relevant in the last decade. Traditionally, message-passing mechanisms were used for communication among the parts of a distributed application where the communications among the application parts were done explicitly by the programmer. Since then, new object distribution techniques have evolved, for instance using Remote Procedure Calls (RPCs) that allow operations to be transparently used regardless of whether the functionality is offered in the local processor or in a remote one.

The object distribution paradigm is probably the most relevant in current industrial applications, and an important example is the CORBA standard [12] which provides a language for the specification of interfaces (IDL, Interface Definition Language) that enables the use of different programming languages in the development of an application. There exist other distribution techniques of higher level coming from CORBA such as CCM (CORBA Component Model), or DDS (Data Distribution Service), but their degree of acceptance in industry is still lower compared to CORBA.

1. This work has been funded in part by the Spanish *Ministry of Science and Technology* under grant number TIC2005-08665-C03-02 (THREAD), and by the IST Programme of the European Commission under project FP6/2005/IST/5-034026 (FRESCOR). This work reflects only the author's views; the EU is not liable for any use that may be made of the information contained herein.

In addition to distribution standards, there are programming languages that allow the development of distributed applications. This is the case of Java (a *de facto* standard) with its specification for distributed systems, Java RMI (Java Remote Method Invocation) [17], based on the distribution of objects. Also, the Ada standard allows distribution through its DSA (Distributed Systems Annex, Annex E) [19], which supports both distribution of objects and RPCs.

This work will focus on analysing the real-time characteristics for distribution within the CORBA and Ada standards. It does not consider Java RMI because the real-time aspects of Java have not been fully addressed yet. RT-CORBA [13] offers the CORBA specification for real-time systems, and although Ada's DSA is not specifically designed for real-time systems, there are works that demonstrate that it is possible to write real-time implementations within the standard [14][7][8]. One goal of this paper is to make a comparative study of the scheduling models offered by these standards for implementing distributed real-time applications, an analysis of some of their implementations from the viewpoint of management of calls to remote resources, and an experimental evaluation on a real-time platform of the response times that can be obtained in remote calls in order to get an idea of the overheads introduced. Another objective of this work is to establish the basis for incorporating the experience acquired in systems programmed in Ada into the world of RT-CORBA.

The evolving complexity of real-time systems has lead to the need for using more sophisticated scheduling techniques, capable of simultaneously satisfying multiple types of requirements such as hard real-time guarantees and quality of service requirements, in the same system. To better handle the complexity of these systems, instead of asking the application to interact directly with the scheduling policies, scheduling services of a higher level of abstraction are being designed, usually based on the concept of resource reservations [2]. The FRESCOR European Union project [3] in which we are participating is aimed at investigating these aspects by creating a contract-based scheduling framework. In [8], some initial ideas were given about the integration of middleware and advanced scheduling services, and in this paper we extend those ideas to address the problem of handling distributed transactions.

The document is organized as follows. Section 2 is dedicated to the presentation of the basic characteristics of the distribution middleware based on RT-CORBA and Ada's DSA, and their implementations. Section 3 analyses in detail the aspects of scheduling, distribution mechanisms, and management of the remote calls proposed in the two standards and their implementations. The evaluation and discussion of the overheads in remote operations for these implementations is dealt with in Section 4. Section 5 proposes the integration of the distribution middleware with the framework for flexible scheduling. Finally, Section 6 draws the conclusions and considers future work.

2 Real-Time Distribution Middleware

This section will describe the scheduling models of RT-CORBA and of the DSA for the execution of remote calls and will discuss how the distributed transaction model

can be supported. Furthermore, the different implementations to be analysed, all of which are open source code, will be briefly introduced.

A distributed transaction is defined as a part of an application consisting of multiple threads executing code in multiple processing nodes, and exchanging messages with information and events through one or more communication networks. In a transaction, events arriving at the system trigger the execution of activities, which can be either task jobs in the processors or messages in the networks. These activities, in turn, may generate additional events that trigger other activities, and this gives way to a chain of events and activities, possibly with end-to-end timing requirements [8]. This model is traditionally used for analysing the response time in real-time distributed applications. We will discuss how this model can be supported by the middleware.

2.1. RT-CORBA Model

The main characteristics of the architecture proposed by RT-CORBA in its specification [12] with respect to scheduling are the following:

- Use of threads as scheduling entities, for which an RT-CORBA priority can be applied and for which there are functions for conversion to the native priorities of the system on which they run.
- Use of two models for the specification of the priority of remote calls (following the Client-Server model): *Client_Propagated* (the invocation is executed in the remote node at the priority of the client, which is transmitted with the request message), and *Server_Declared* (all the requests to a particular object are executed at a priority preset in the server). In addition, it is possible for the user to define priority transformations that modify the priority associated with the server. This is done with two functions called *inbound* (which transforms the priority before running the server's code) and *outbound* (which transforms the priority with which the server makes calls to other remote services).
- Definition of *Threadpools* as mechanisms for managing remote requests. The threads in the pool may be preallocated, or can be created dynamically. There may be several groups of threadpools, each group using a specific priority band.
- Definition of *Priority-Banded Connections*. This mechanism is proposed for reducing priority inversions when a transport protocol without priorities is used.

The specification of RT-CORBA incorporates a chapter dedicated to dynamic scheduling, which basically introduces two concepts:

- The possibility of introducing other scheduling policies in addition to the fixed priority policy, such as, EDF (Earliest Deadline First), LLF (Least Laxity First), and MAU (Maximize Accrued Utility). The scheduling parameters are defined as a container that can contain more than one simple value, and can be changed by the application dynamically.
- The *Distributable Thread* that allows end-to-end scheduling and the identification of *Scheduling Segments* each one of which can be run on a processor. This concept is similar to the distributed transaction presented in [8].

RT-CORBA does not consider explicitly the possibility of passing scheduling parameters to the communications networks.

2.2. Ada DSA Model

Ada DSA does not have any mechanism for transmission of priorities and so its implementation is left up to the criterion of the implementation. The specification requires support for executing concurrent remote calls and for waiting until the return of the remote call. The communication among active partitions is carried out in a standard way using the Partition Communication Subsystem (PCS).

The concurrency and the real-time mechanisms are supported by the language itself with tasks, protected types and the services specified in Annex D. In [4], a mechanism for handling the transmission of priorities in the DSA is proposed. This mechanism is in principle more powerful than that of RT-CORBA, as it allows total freedom in the assignment of priorities both in the processors and in the networks used.

Ada included in its latest revision the scheduling policies EDF and Round Robin as part of its Real-Rime Systems Annex (Annex D). Nevertheless, it does not contemplate the existence of distributed transactions. Like RT-CORBA, Ada DSA does not consider the possibility of passing scheduling parameters to the communications networks.

2.3. Implementations under study

This work analyses and assesses the following implementations of RT-CORBA and the DSA:

- TAO [18] is an open source implementation of RT-CORBA that has been evolving for several years. The applications are programmed in C++ and the version we have used (1.5) runs on Linux and TCP/IP. It is offered as an implementation of the complete specification.
- PolyORB [15][20] is presented as a “schizophrenic” middleware that can support distribution with different personalities such as CORBA, RT-CORBA, or DSA. It is distributed with the GNAT compiler [1] and in principle it is envisaged for applications programmed in Ada. The version used (2007) supports CORBA and some basic notions of RT-CORBA (priorities and their propagation), and allows distribution through the DSA although it does not allow specifying scheduling parameters. The execution platform is Linux and TCP/IP.
- GLADE [14] is the original implementation of the DSA offered by GNAT [1] to support the development of distributed applications with real-time requirements. The scheduling is done through fixed priorities and implements two policies for distribution of priorities in the style of RT-CORBA (Client Propagated and Server Declared). The 2007 version is used, and once again the execution platform is Linux and TCP/IP.
- RT-GLADE is a modification of GLADE that optimizes its real-time behaviour. There are two versions: in the first one [7], free assignment of priorities in remote

calls is allowed (both in the processors and in the communication networks). The second version [8] proposes a way of incorporating distributed transactions into the DSA and giving support to different scheduling policies in a distributed system. The execution platform is MaRTE OS [9] and the network protocol is RT-EP [10]. This communication protocol is based on token passing in a logical ring over standard Ethernet, and it supports three different scheduling policies: fixed priorities, sporadic servers, and resource reservations through contracts [2][3].

3 Analysis of Distribution Middleware Implementations

The objective of this section is to analyse the scheduling aspects of the mechanisms for management of remote calls used by the implementations of RT-CORBA or DSA to support their respective specifications. It also discusses the properties of the solutions adopted and proposes some improvements that could be made both in the standards and in their implementations.

3.1. Implementations of RT-CORBA and DSA

From the viewpoint of management of remote calls, TAO defines several elements that can be configured [16]:

- Number of ORBs. The ORB is the management unit of the calls to a service. There may be several or only one, given that each ORB can accept requests from different parts of the application.
- The strategy of the concurrency server. Two models are defined: *Reactive*, in which a thread is executed to provide service to multiple connections; and *thread-per-connection*, in which the ORB creates a thread to serve each new connection.
- The threadpools. Two types of thread groups are defined with two different behaviours. In the *ORB-per-Thread* model each thread has an associated ORB that accepts and processes the services requested. In the *Leader/Followers* model the user can create several threads and each ORB will select them in turns so they await and process new requests arriving from the network.

For the management of remote calls, PolyORB defines the following configurable elements [15]:

- ORB tasking policies. Four policies are defined:
  - *No_Tasking*: the ORB does not create threads and uses the environment task to process the jobs
  - *Thread_Pool*: a set of threads is created at start-up time; this set can grow up to an absolute maximum, and unused threads are removed from it if its size exceeds a configurable intermediate value.
  - *Thread_per_Session*: a thread is created for each session that is opened
  - *Thread_per_Request*: a thread is created for each request that arrives and is destroyed when the job is done

- Configuration of the tasking runtimes. It is possible to choose among a Ravenscar-compliant, no tasking, or full tasking runtime system.
- ORB control policies. Four policies are defined that affect the internal behaviour of the middleware:
  - *No Tasking*: a loop monitors I/O operations and processes the jobs
  - *Workers*: all the threads are equal and they monitor the I/O operations alternatively
  - *Half Sync/Half Async*: one thread monitors the I/O operations and adds the requests to a queue, and the other threads process them
  - *Leader/Followers*: Several threads take turns to monitor I/O sources and then process the requests once arrived. However, if RT-CORBA is in use, the selected thread will add the request to an intermediate queue where another thread will process it at the proper priority.

The implementation of the DSA carried out in GLADE [14] defines a group of threads to process the requests with similar parameters to those of PolyORB in terms of the number of threads (minimum number of threads created at start-up time, stable value and absolute maximum), and uses another two intermediate threads for the requests; one awaits the arrival of requests from the network, and the other one processes these requests and selects one of the threads of the group to finally process the job.

The modifications made to GLADE to obtain the first version of RT-GLADE [7] eliminated one of the intermediate threads, so that there was a thread waiting for requests arriving from the net, which in turn activated one of the threads of the group to carry out the job. In the second version of RT-GLADE [8], an API was provided to allow an explicit configuration of the threads that execute the jobs, and they are designed to wait directly on the net. This is done through the definition of communication *endpoints* which handle the association with the remote thread and support the scheduling parameters for the network. These parameters, that can be complex, are associated with the appropriate entity when a distributed transaction is installed and do not need to be transmitted each time the remote service is called.

TAO, PolyORB, and GLADE all use the priority assignment policies defined in RT-CORBA. In contrast, in the first version of RT-GLADE [7] free assignment of priorities is allowed for the remote services and for the request and reply messages. This approach enables the use of optimization techniques in the assignment of priorities in distributed systems.

In the second version of RT-GLADE [8], the definition of the connection endpoints allows the programming of distributed transactions, which are identified just by specifying a small number at the beginning of the transaction. Moreover, the transaction is executed with the scheduling parameters associated to its threads and messages. This concept is similar to the distributable thread of RT-CORBA, except that this specification never takes the network scheduling into account. TAO implements this part of the dynamic scheduling of RT-CORBA, in which dynamic changing of the scheduling parameters of a scheduling segment is permitted [5].

In this work, we have made a prototype porting of PolyORB to the MaRTE OS [9] real-time operating system and we have adapted it to the RT-EP real-time network protocol [10]. The personality of CORBA (PolyORB-CORBA) allows the use of the control policies of the ORB defined in PolyORB. The DSA personality of PolyORB does not currently allow choosing among different control policies. For this personality (PolyORB-DSA), a basic version of the scheduling defined in [8] has been implemented over our real-time platform to obtain results comparable to those of RT-GLADE.

3.2. Discussion

Based on the analysis above, this subsection discusses some objectives that the real-time distribution middleware must pursue, and proposes solutions or extensions that the standards and/or the implementations should incorporate.

- Allow a schedulability analysis of the complete application. Although the middleware is executed in the processor, in many cases the timing behaviour of the networks has a strong influence on the overall response times, and therefore the networks should be scheduled with appropriate techniques [6]. The middleware should have the ability to specify the scheduling parameters of the networks through suitable models. RT-GLADE could be used as a reference [8].
- Transactions or distributable threads. In agreement with the previous point, the transactions or distributable threads should incorporate all the information about scheduling in the processors and networks, either in the model proposed by RT-CORBA or in the one proposed in RT-GLADE [8].
- Control of remote calls. The task models implemented in TAO and PolyORB can be used as a reference, adding an extra case in which there is one dedicated thread per kind of request, directly waiting on the net (as in the second version of RT-GLADE). The latter case can be useful in flexible scheduling environments when threads execute under contracts and the cost of negotiating or changing contracts is very high. In the case when there are intermediate threads for managing remote calls (GLADE, RT-GLADE or PolyORB) it is important to control their scheduling parameters. This is also the case of groups of threads in which threads can execute with different parameters each time.
- Allow the free assignment of scheduling parameters. This is the approach used in RT-GLADE. In RT-CORBA there is a specification for static real-time systems, and an extension for dynamic real-time systems (see Section 3 in [13]). The specification for static systems imposes restrictions on the assignment of priorities, but these restrictions are removed in the specification for dynamic systems, which allows implementations to define scheduling policies.

4 Evaluating Distribution Middleware Implementations

The objective of this section is to provide an idea about the predictability and the overhead introduced by the analysed implementations in a distributed application, but not to make a straight comparison among them, as they are of different nature.

In this work, we have made a prototype porting of PolyORB to the MaRTE OS [9] real-time operating system and we have adapted it to the RT-EP real-time network protocol [10]. The personality of CORBA (PolyORB-CORBA) allows the use of the control policies of the ORB defined in PolyORB. The DSA personality of PolyORB does not currently allow the definition of any particular control policy. For this personality (PolyORB-DSA), a basic version of the scheduling defined in [8] has been implemented over our real-time platform. Furthermore, GLADE 2007 has been modified to support the mechanisms included in the second version of RT-GLADE.

In flexible scheduling environments threads are executed under contracts and the cost of negotiating or changing them could be very high for the system. To minimize context switches and therefore fit those requirements, RT-GLADE uses dedicated threads that wait for the event arrivals and then process the received events. The *Leader/Followers* pattern uses a similar concept, with threads that perform both communication and processing roles, thus minimizing context switches. This pattern is the one that is most similar to the RT-GLADE approach, and consequently it will be used in this evaluation both for TAO and PolyORB.

A hardware platform consisting of two 800-MHz AMD Duron processors and a dedicated 100-Mbps Ethernet has been used. The following two software platforms have also been used:

- Linux kernel 2.6.10 with TCP/IP to evaluate the implementations of TAO (version 5.5), PolyORB (version 2.3) with CORBA personality and GLADE (version 2007).
- MaRTE OS 1.7 with RT-EP [10] to evaluate PolyORB-CORBA (version 2.3), PolyORB-DSA (version 2.3) and RT-GLADE (adapted from GLADE 2007).

The tests will measure the execution time of a remote operation that adds two integers and returns the result. The measurement is carried out from the time when the call is made until the response is returned. This operation will be carried out in two modes: alone, and with four other clients carrying out the same operation, but at a lower priority. The objective is not to obtain exhaustive measurements of the platform, but to get a rough idea of the performance (predictability and overheads) that can be achieved with the middleware. In all the tests the operation to be evaluated is executed 10,000 times, and the average, maximum, and minimum times are evaluated, together with the standard deviation and the relative frequency of time values that are within a deviation from the maximum of 10% of the difference between the maximum and average values.

Table 1 and Table 2 show the results of the measurements taken with the Linux platforms, using the middleware configurations that introduce the least overhead. For the case of a single client in TAO the reactive concurrency model with a single thread in the group has been used. In PolyORB the model with full tasking without internal threads has been used for the experiment with one client. For the five-client case both in TAO and in PolyORB a configuration of a group of 5 threads with a *Leader/ Followers* model has been used. In GLADE, a static group of threads equal to the number of clients is defined. The priority specification model for TAO, PolyORB and

Table 1. Measurements in Linux for one client (times in μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
TAO	998	1380	914	75	0.06
PolyORB-CORBA	1424	4302	1189	373	0.01
GLADE	415	3081	340	261	0.02
Stand-alone network	129	678	118	40	0.12

Table 2. Measurements in Linux for the highest priority client, five clients (times in μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
TAO	1371	6376	889	356	0.02
PolyORB-CORBA	5399	11554	1593	1050	0.02
GLADE	1700	5953	595	496	0.12

GLADE was client propagated. In order to make the middleware overhead measurements more comparable, the temporal cost of using the net is also evaluated. Thus, Table 1 includes the average, maximum and minimum times for the case when a message is sent and a response is received; the program on the server side answers immediately upon reception.

In the results obtained for one client in Linux, it can be observed that GLADE achieves better average and minimum response times than TAO and PolyORB, which can be explained because it has a lighter code. The maximum times obtained for PolyORB and GLADE in the case of one client are much higher than the average times compared to TAO. The average numbers for one and five clients show large differences in PolyORB and GLADE, while in TAO they are relatively similar. We can conclude that this configuration of TAO makes a better management of the priorities and the queues on this platform.

Table 3 and Table 4 show the results of the measurements carried out over the three implementations on the MaRTE OS/RT-EP platform. The configuration of PolyORB-CORBA is the same as for Linux. The PolyORB-DSA configuration creates a task explicitly to attend the remote requests. The group of threads for RT-GLADE is configured to be equal to the number of clients. As for RT-EP, the parameter corresponding to the delay between arbitration tokens is set to a value of 150 μs. This value limits the overhead in the processor due to the network. A simple transmission in the network is also evaluated for the same reason as in the case of Linux (see Table 3).

From the results obtained in the evaluation on the real-time platform, it can be observed that, firstly, the network protocol has a greater latency and it makes the times of a simple round-trip transmission higher than in Linux; the trade-off is that this is a predictable network with less dispersion among the values of the measurements. Furthermore, the minimum and average times of RT-GLADE for one client are also greater than those of GLADE over Linux, although the maximum time remains within a bound indicating a much lower dispersion. An important part of the response times obtained for RT-GLADE is due to the network, but is also due to the operating system

Table 3. Measurements in MaRTE OS for one client (times in μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	2997	3012	2770	6	0.01
PolyORB-DSA	4117	4487	3835	300	42.50
RT-GLADE	1080	1151	955	23	0.03
Stand-alone network	959	964	707	3	0.01

Table 4. Measurements in MaRTE OS for the highest priority client, five clients (times in μs)

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	3527	6566	2748	727	0.11
PolyORB-DSA	4516	5299	3531	320	0.02
RT-GLADE	1000	1462	896	27	0.06

and the dynamic memory manager used [11] (to make the timing predictable). If we observe the times of RT-GLADE for five clients, we can see that only the minimum time is worse than in GLADE, although with less difference; in contrast the average time and specially the maximum are now clearly better. The increase in the maximum times of RT-GLADE with respect to the case of one client is reasonable and can be justified by the blocking times that can be suffered both in the processor and in the network.

In the measurement of the times of PolyORB-CORBA over MaRTE OS we have found a great disparity of the measurements for five clients depending on the priorities used in them. This can be explained because of the architecture used to implement the *Leader/Followers* model. This is a part which could be improved by using an implementation model similar to TAO. In any case, the measurements reflected in Table 4 for PolyORB-CORBA with five clients have been obtained in a best-case scenario in which the low-priority clients are not preempted by any of the threads in the thread pool. Referring to PolyORB-DSA, the response times obtained are comparable to those of PolyORB-CORBA, but with higher predictability.

As a consequence of the response times of PolyORB over MaRTE OS, it is again shown, by comparing the results with those of RT-GLADE, that the pure implementation of the DSA can be much lighter than that of RT-CORBA. Comparing the tests of PolyORB-CORBA for one and for five clients it can be seen that there is an important difference between the minimum and maximum times for five clients, which is due to the priority inversion introduced by the intermediate tasks.

5 Integration of the Distribution Middleware with a Contract-Based Scheduling Framework

The FRESCOR (Framework for Real-time Embedded Systems based on COntRacts) EU project [3] has the objective of providing engineers with a scheduling framework that represents a high-level abstraction that lets them concentrate on the specification

of the application requirements, while the system transparently uses advanced real-time scheduling techniques to meet those requirements. In order to keep the framework independent of specific scheduling schemes, FRESCOR introduces an interface between the applications and the scheduler, called the service contract. Application requirements related to a given resource are mapped to a contract, which can be verified at design time by providing off-line guarantees, or can be negotiated at runtime, when it may or may not be admitted. As a result of the negotiation a virtual resource is created, representing a certain resource reservation. The resources managed by the framework are the processors, networks, memory, shared resources, disk bandwidth, and energy; additional resources could be added in the future.

Careful use of virtual resources allows different parts of the system (whether they are processes, applications, components, or schedulers) to use budgeting schemes. Not only can virtual resources be used to help enforce temporal independence, but a process can interact with a virtual resource to query its resource usage and hence support the kinds of algorithms where execution paths depend on the available resources.

When distribution middleware is implemented on operating systems and network protocols with priority-based scheduling, it is easy to transmit the priority at which a remote service must be executed inside the messages sent through the network. However, this solution does not work if more complex scheduling policies, such as the FRESCOR framework, are used. Sending the contract parameters of the RPC handler and the reply message through the network is inefficient because these parameters are large in size. Dynamically changing the scheduling parameters of the RPC handler is also inefficient because dynamically changing a contract requires an expensive renegotiation process.

The solution proposed in [8] consisted in explicitly creating the network and processor schedulable entities required to establish the communication and execute the remote calls. The contracts of these entities are negotiated and created before they are used. They are then referenced with a short identifier that can be easily encoded in the messages transmitted. For identifying these schedulable entities the transactional model is used and the identifier, called an Event_Id, represents the event that triggers the activity executed by the schedulable entity.

In the current FRESCOR framework, support for the transactional model is being built. A tool called the Distributed Transaction Manager (DTM) is a distributed application responsible for the negotiation of transactions in the local and remote processing nodes in a FRESCOR system that implements the contract-scheduling framework. Managing distributed transactions cannot be done on an individual processing node because it requires dynamic knowledge of the contracts negotiated in the other nodes, leading to a distributed consensus problem. The objective of the Distributed Transaction Manager is to allow the remote management of contracts in distributed systems, including capabilities for remote negotiation and renegotiation, and management of the coherence of the results of these negotiation processes. In this way, FRESCOR provides support for distributed global activities or transactions

consisting of multiple actions executed in processing nodes and synchronized through messages sent across communication networks.

The implementation of the DTM contains an agent in every node, which listens to messages either from the local node or from remote nodes, performs the requested actions, and sends back the replies. In every node there is also a DTM data structure with the information used by the corresponding agent. Part of this information is shared with the DTM services invoked locally from the application threads. This architecture could benefit from the presence of a distribution middleware, by making the agents offer operations that could be invoked remotely, thus simplifying the current need for a special communications protocol between the agents.

The current version of the transaction manager limits its capabilities just to the management of remote contracts. In the future, the DTM should also provide a full support for the transactional model, integrated with the distribution middleware. For this purpose the following services would need to be added to it:

- Specification of the full transaction with identification of its activities, remote services and events, and contracts for the different resources (processors and networks).
- Automatic deployment of the transaction in the middleware. This would require:
  - choosing unused Event_Ids for the transaction events
  - choosing unused ports in the involved nodes, for the communications
  - creating send endpoints for the client-side of the communications, using the desired contracts and networks
  - creating receive endpoints for the reception of the reply in the client-side of the communications, using the desired networks, ports, and event ids.
  - creating the necessary RPC handlers with their corresponding contracts
  - creating the receive endpoints of the server-side of the communications using the desired contracts and networks
  - creating the send endpoints of the server-side of the communication using the desired contracts and networks.

All this deployment would be done by the DTM from the information of the transaction, which could be written using a suitable deployment and configuration language. After this initialization, the transaction would start executing, its RPCs would be invoked and the middleware would automatically direct them through the appropriate endpoints and RPC handlers almost transparently. We would just specify the appropriate event ids.

With the described approach we would achieve a complete integration of the distribution middleware and the transactional model in a system managed through a resource reservation scheduler.

6 Conclusions and Future Work

The work presented here reports an analysis and evaluation of some implementations of distribution middleware from the viewpoint of their suitability for the implementation of real-time systems. Specifically, the following aspects have been highlighted: the way remote calls are managed, the mechanisms for establishing the scheduling parameters, and the importance of giving support to the transactions or distributable threads.

The time measurements have been carried out over Linux as the native operating system of the middleware analysed, and over a real-time platform based on the MaRTE operating system and the RT-EP real-time network protocol, to which PolyORB has been ported in this work. In the measurements obtained, it can be observed that the implementations of Ada's DSA are lighter than the implementations of RT-CORBA. This suggests that Ada is a good option for programming distributed systems, and that it could find its niche in medium-sized embedded distributed real-time systems. The measurements on the real-time platform also show that the predictability has a cost in terms of overhead in the network and in memory management.

Furthermore, new mechanisms for contract-based resource management in a distributed real-time system have been identified, and the necessity to integrate the distribution middleware with them has been described, together with some ideas on future work needed to support this integration.

Our work will continue with experimentation on the PolyORB real-time platform that we already have, given our experience in Ada and in GLADE. The objective will be to progress with the improvement of specific real-time aspects over this platform both for the DSA and for RT-CORBA, and to integrate the distributed transaction model along with their managers and the new contract-based scheduling mechanisms for processors and networks using the ideas described in this paper. The ultimate goal would be to make proposals for inclusion in the corresponding standards and implementations.

References

1. Ada-Core Technologies, The GNAT Pro Company, http://www.adacore.com/
2. M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J.M. Drake, G. Fohler, P. Gai, M. González Harbour, G. Guidi, J.J. Gutiérrez, T. Lennvall, G. Lipari, J.M. Martínez, J.L. Medina, J.C. Palencia, and M. Trimarchi. "FSF: A Real-Time Scheduling Architecture Framework". Proc. of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2006, San Jose (CA, USA), 2006.
3. FRESCOR project web page: http://frescor.org
4. J.J. Gutiérrez, and M. González Harbour. "Prioritizing Remote Procedure Calls in Ada Distributed Systems". Proc. of the 9th International Real-Time Ada Workshop, ACM Ada Letters, XIX, 2, pp. 67–72, June 1999.
5. Y. Krishnamurthy, I. Pyarali, C. Gill, L. Mgeta, Y. Zhang, S. Torri, and D.C. Schmidt. "The Design and Implementation of Real-Time CORBA 2.0: Dynamic Scheduling in

TAO". Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04), Toronto (Canada), May 2004.

6. J. Liu. "Real-Time Systems". Prentice Hall, 2000.
7. J. López Campos, J.J. Gutiérrez, and M. González Harbour. "The Chance for Ada to Support Distribution and Real Time in Embedded Systems". Proc. of the International Conference on Reliable Software Technologies, Palma de Mallorca, Spain, in LNCS, Vol. 3063, Springer, June 2004.
8. J. López Campos, J.J. Gutiérrez, and M. González Harbour. "Interchangeable Scheduling Policies in Real-Time Middleware for Distribution". Proc. of the 11th International Conference on Reliable Software Technologies, Porto (Portugal), in LNCS, Vol. 4006, Springer, June 2006.
9. MaRTE OS web page, http://marte.unican.es/
10. J.M. Martínez, and M. González Harbour. "RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet". Proc. of the 10th International Conference on Reliable Software Technologies, York (UK), in LNCS, Vol. 3555, Springer, June 2005.
11. M. Masmano, I. Ripoll, A. Crespo, and J. Real. "TLSF: A New Dynamic Memory Allocator for Real-Time Systems". Proc of the 16th Euromicro Conference on Real-Time Systems, Catania (Italy), June 2004.
12. Object Management Group. "CORBA Core Specification". OMG Document, v3.0 formal/ 02-06-01, July 2003.
13. Object Management Group. "Realtime CORBA Specification". OMG Document, v1.2 formal/05-01-04, January 2005.
14. L. Pautet, and S. Tardieu. "GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems". Proc. of the 3rd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing, (ISORC'00), Newport Beach, USA, March 2000.
15. PolyORB web page, http://polyorb.objectweb.org/
16. I. Pyarali, M. Spivak, D.C. Schmidt, and R. Cytron. "Optimizing Thread-Pool Strategies for Real-Time CORBA". Proc. of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 2001.
17. Sun Developer Network, http://java.sun.com
18. TAO web page, http://www.cs.wustl.edu/~schmidt/TAO.html
19. S. Tucker Taft, Robert A. Duff, Randall L. Brukardt, Erhard Ploedereder, and Pascal Leroy (Eds.). "Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1". LNCS 4348, Springer, 2006.
20. T. Vergnaud, J. Hugues, L. Pautet, and F. Kordon. "PolyORB: a Schizophrenic Middleware to Build Versatile Reliable Distributed Applications". Proc.of the 9th International Conference on Reliable Software Technologies, Palma de Mallorca (Spain), in LNCS, Vol. 3063, June 2004.