Infrastructure for Virtual Enterprises in Large-Scale Open Systems

Joni Fraga[1], Ricardo J. Rabelo[1], Frank Siqueira[2], Carlos B. Montez[1], Rômulo S. Oliveira[1]
Department of Automation and Systems (DAS) [1] *and Department of Informatics and Statistics (INE)* [2]
Federal University of Santa Catarina (UFSC), Florianópolis, Santa Catarina, Brazil
{fraga, rabelo, montez, romulo}@das.ufsc.br[1], *frank@inf.ufsc.br*[2]

Abstract

This paper presents SALE, a middleware platform for mission-critical applications built upon the CORBA architecture and its extensions for real-time (RT-CORBA), fault tolerance (FT-CORBA) and security (CORBASec). The SALE platform provides different reliability, security and real-time properties, which are specified by applications as quality of service (QoS) requirements. Preliminary results obtained through the implementation of this platform allow us to use it as an infrasctructure for business-to-business applications. This infrasctructure provides a multi-agent system for production scheduling of virtual enterprises, in which timing requirements must be enforced in order to execute a manufacturing process cooperatively.

1. Introduction

The temporary association of enterprises that collaborate towards obtaining a given product leads to the creation of so-called *virtual enterprises*. Virtual enterprises are formed when originally independent companies unite in pursuit of a common goal and exchange information in order to coordinate their actions [1]. Such a scenario is illustrated by Figure 1.

The manufacturing process of virtual enterprises involves a vast array of technologies, including *hardware* and *software* resources that are distributed among large-scale computing systems. The virtual enterprise paradigm implies that the enterprises must use communication mechanisms, such as EDI (*Electronic Data Interchange*) and *Workflow*, in order to exchange information and to synchronize electronically their manufacturing processes. High-level services, such as support for cooperative product development and real-time monitoring of order deliveries, require extremely robust, safe and efficient infrastructures in order to be used by enterprises. At the administrative level, a series of tools must be available so that the enterprises might cooperate effectively. The enterprises involved in the manufacturing process need tools for activity scheduling to coordinate their actions, and need collaboration tools to cooperate effectively.

Figure 1. Business process in a virtual enterprise

From what has been shown above, it is clear that some production stages require real-time guarantees, especially in the manufacturing process and in the cooperative project. Other stages require high reliability, especially in business-to-business and business-to-client transactions. Moreover, security mechanisms must be employed during exchange of information for developing cooperative projects and during financial transactions, in order to avoid malicious external interference.

The complexity of the technologies employed by virtual enterprises, along with the heterogeneity of large-scale systems, require a computer support that simplifies the use of such technologies. This support must enforce security, reliability and real-time constraints during interactions among enterprises. In order to do so, the support must provide applications with mechanisms for specification of their quality of service (QoS) constraints.

The remainder of this paper is organised as follows. Section 2 presents SALE – a support for large-scale open systems for virtual enterprises, built upon the CORBA platform, using its extensions for real-time, fault tolerance and security. Section 3 describes SC^2 – a manufacturing scheduling system for virtual enterprises developed using the SALE platform. Finally, Section 4 analyzes the obtained results, and Section 5 presents our conclusions and perspectives for future work.

2. Support for Applications in Large-Scale Open Systems: The SALE Project

The SALE Project is aimed at developing a platform whose architecture integrates services with different kinds of guarantees. Such platform, suited to new applications which are categorised as globally distributed systems and deal with different kinds of data, must include some functionalities and characteristics which contribute to the flexibility and adaptation of the support, taking into account the needs of the application. To meet such requirements, it is necessary:

- To provide solutions that are suitable to the complexity and heterogeneity of large-scale systems;
- To make extensive use of open standards, which ensure the availability of a broad array of technologies;
- To provide a suitable and secure environment for the specialization of support services.

The SALE architecture is based on the CORBA (*Common Object Request Broker Architecture*) standards, proposed by the OMG (*Object Management Group*) [2]. With CORBA, the OMG has defined a standard for the communication between objects in a distributed and heterogeneous environment, offering advantages such as interoperability, portability and reuse of application code.

2.1. The SALE Architecture

The SALE architecture, illustrated by Figure 2, is aimed mainly at integrating mechanisms that provide different reliability, security and real-time properties in the interactions among applications. These different attributes are to be specified as quality of service (QoS) requirements. Support services are specified according to such QoS requirements, involving different mechanisms both on the execution platform and on the underlying layers. The mechanisms required for QoS specification are made available from a unique interface, the *QoS object* (Figure 2), which will be described in more detail in the next section.

The reliability, security and real-time requirements of the application are enforced by using the *Fault-Tolerant CORBA* (FT-CORBA) [3], *CORBA Security* (CORBASec) [4] and *Real-Time CORBA* (RT-CORBA) [5] object services, respectively. Additionally, depending on the QoS requirements of the application, underlying group communication, secure communication and real-time communication supports can also be used, as shown by Figure 2. At the higher level, the *CORBA Notification Service* [6] provides event-based communication with QoS constraints. Therefore, the SALE platform is fully compliant with the CORBA Architecture and with the Common Object Services defined by the OMG.

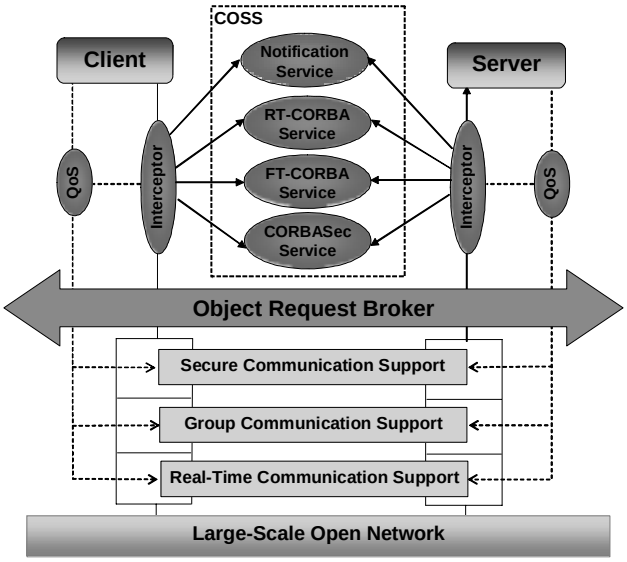

Figure 2. The SALE Architecture

CORBA *interceptors*(1) are mechanisms used for adaptation of the middleware platform according to the needs of applications. The interception in each method invocation through the ORB, allows for imposing control policies on the method execution in a transparent way. In SALE, the configuration best adapted to the QoS requirements of the application is chosen among the available ORBs and object services. Selection through such requirements influences both the way the local system executes the application and the way the communication is performed.

The ORBs and the CORBA object services which provide support for real-time, reliability and security are instantiated separately. Therefore, the platform will only deal with one requirement at a time. Throughout the development of the project, we have made efforts to harmonize those requirements that show no conceptual incompatibility. This process must follow the guidelines of the OMG, which has shown considerable concern about making RT-CORBA, FT-CORBA and CORBASec compatible [4].

In the following sections, we present our experiences obtained during the development of SALE related to the enforcement of timing constraints. Our efforts regarding fault-tolerance and security have been directed towards implementing these specifications separately in the *GroupPac* [7, 8] and *JaCoWeb* [9, 10] projects. Information on *JaCoWeb* and *GroupPac* can be obtained at the following URLs: http://www.lcmi.ufsc.br/grouppac and http://www.lcmi.ufsc.br/JaCoWeb.

(1) Interceptors are objects logically interposed in a client-to-server invocation path, aimed at diverting calls transparently, activating associated service objects, and controlling the application [2].

2.2. Mechanisms for QoS Specification and Enforcement

QoS specification mechanisms must be very flexible in order to express QoS requirements using application-level concepts intelligible at different application domains.

The SALE platform provides both static and dynamic mechanisms for the specification of QoS. These are the QSL language and the QoS Object, respectively.

The QSL language (Quality Specification Language) [11], allows the QoS requirements of an application to be expressed statically. QoS requirements can be associated with individual calls, with all the calls to the same method or with all calls to an object. QSL *scripts* specify the way the different parameter sets are mapped into system resources. In addition, QSL provides QoS specifications for RT-CORBA, FT-CORBA, CORBA*Sec* and for the Notification Service, which can be inherited and specialised by applications.

The *QoS Objects* used by SALE are generated based on the QSL specification associated to a particular application. As shown in Figure 2, QoS objects are located at the application level, together with the client and the server objects, allowing the computing support to be configured during binding time in order to meet the application requirements. This approach is compatible with the RT-CORBA, FT-CORBA and CORBA*Sec* models, in which the object services and ORB mechanisms that act during the execution time in client-server interactions are created when bindings are established between objects. The QoS objects are consulted by interceptors during the invocation in order to obtain the values of QoS parameters, so that QoS requirements are enforced while the distributed application is running. QoS objects also allow us to modify dynamically the value of QoS parameters (for example, the CORBA priority of methods) by using the IDL interface shown in Figure 3.

Meeting the QoS requirements of the application will depend on the availability of resources from the execution platforms and the communication support. The QoS parameters of the application are mapped into these underlying resources, which are allocated in order to ensure to the application the desired quality of service.

2.3. SALE and Real-Time CORBA

The RT-CORBA specification [5] provides the ORB with mechanisms for resource management and for enforcing predictable behavior.

Scheduling policies, which deliver predictable behavior in real-time systems, are implemented by assigning priorities to threads, the schedulable entities in the system. The *CORBA priority* is a global priority independent from the execution platform, which can cross

```
// From the CORBA Property Service
struct Property{
  string property_name;
  any    property_value;
};
typedef sequence<Property> Properties;

// SALE IDL
module SALE {
  interface Qos {
    exception ParamEx { Properties
                        qos_params; };
    exception ParseEx { short num_line;
                        string qsl_expr; };
    attribute Properties myQos;
    void readQosSpec (in string qsl_spec)
       raises (ParamEx, ParseEx);
    void setParam (in string name, in any val)
      raises (ParamEx);
    void getParam (in string name,
                   out any val)
       raises (ParamEx);
    void configQos() raises (paramEx);
  };
  ...
};
```

Figure 3. SALE's QoS object IDL

ORB boundaries within an invocation message. This priority is used by servers for ordering requests. The RT-CORBA specification introduces two priority models: *client-propagated* and *server-declared*. In the client-propagated model, a CORBA priority is specified by the client and propagated with an invocation, and is used to define the priority of the thread in which the request will be executed. In the server-declared model, each CORBA object has a predefined priority, which is informed to clients within its object reference.

When the client-propagated model is used, it is possible to perform priority transforms in order to modify the value of priorities during an invocation. This mechanism resembles an interceptor, acting on the server-side before and after processing an invocation, but being only able to transform the priority of method invocations. Independently from the priority model used by the application, every CORBA priority must be mapped by the ORB into operating system native priorities before the execution. RT-CORBA specifies an interface for executing this mapping, and establishes that every RT-ORB must provide a default mapping scheme.

Threadpools and *invocation queues* can be handled directly by applications, and can be used to improve predictability. Threadpools can be created to process method invocations received by a server, while invocation queues can be associated to threadpools.

RT-CORBA also increases the portability of applications by providing *thread management interfaces*, which allow threads to be controlled independently of the multithreading support and the scheduling mechanism provided by the operating system.

During an *explicit binding* of objects, a series of actions must be performed to interconnect these objects, such as locating the destination objects and initiating the data structures used for the communication. On the client side, a binding can select an appropriate transport protocol and a timeout value for blocking the client. On the server side, the binding provides the allocation of the necessary resources for the execution of the requests, such as threads and queues.

All mechanisms defined by the RT-CORBA specification consider the adoption of fixed-priority scheduling, but also define mechanisms that can be used for implementing dynamic priority scheduling.

The Real-Time Invocation Model. In SALE, the client's timing requirements are implemented by using CORBA priorities that can reflect deadlines or other time constraints. That is, the scheduling policies that interest project SALE are client-propagated.

Figure 4 presents the Real-Time Invocation Model used by SALE, which is based on the RT-CORBA specification. The scheduling approach is implemented by an object service called the Scheduling Service that runs at all server nodes. Interceptors at the server side are used to redirect the requests to the Scheduling Service in a transparent way. This service is responsible for the implementation of the scheduling policies that act on the invocation of methods at the server. A thread pool is pre-allocated by the server in order to execute methods requested by its clients, increasing the performance of method executions.

The real-time communication support depends on the lower-level services provided by the network. Due to the non-deterministic approach adopted, transient overheads may occur, which are treated using adaptive techniques only in the server nodes [12]. The client assigns an absolute deadline or a global priority to the method invocation before sending the request, thus characterizing a real-time invocation. When absolute deadline values are used, it is necessary to use a global time basis, which makes possible a coherent vision of these time values in the whole system. In [13], a *Global Time Sevice* extends the specifications of the CORBA Time Service, providing a global time basis. The Global Time Service can be implemented in a broad and scattered system, by using GPS receivers.

In the prototype implemented, the Real-Time Invocation Model considers each invocation as fully executed, even after the deadline. There are no discards or abortions in the execution of requests, although theoretically a method with a deadline does not provide maximum benefit when it is executed after the deadline. Therefore, step (iv) in Figure 4 is used only to implement adaptive policies that dynamically assign priority values based on a history of deadline misses. In this prototype,

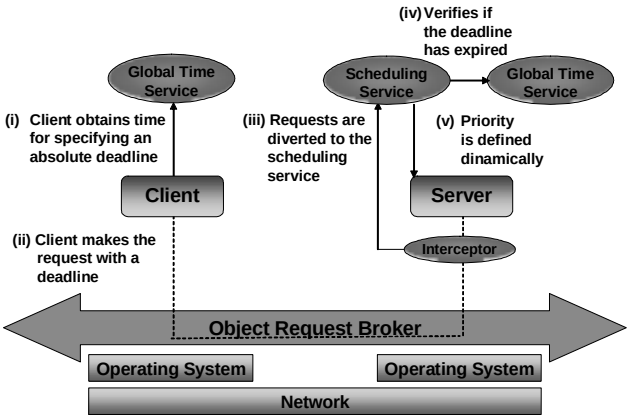

Figure 4. Real-Time Invocation Model

according to what we will discuss in section 3.2, the deadline value (absolute deadline) is defined using the server clock, based on an interval (relative deadline) propagated by the client. This solution eliminates the need for a global time service and GPS receivers.

In our prototype implementation of the Real-Time Invocation Model, we have used the TAO ORB – a high performance ORB for real-time systems developed at Washington University at Saint Louis [14], intended for deterministic environments. TAO had a great influence on the development of the RT-CORBA standard approved by the OMG. We have extended the TAO ORB, adding some mechanisms in an attempt to fully comply with the RT-CORBA specification. For example, support for request interceptors was added because they were not implemented in the earlier versions of TAO.

2.3. The Notification Service

The Notification Service [6] is an extension of the CORBA Event Service. Both services implement a decoupled asynchronous communication mechanism for the exchange of events. They are built around event channels, which are used by publishers (event suppliers) to notify subscribers (event consumers) of the occurrence of events. By using an event channel, publishers and subscribers do not need to know each other to exchange events, and do not have to be active at the same time. Events can be typed or untyped (in this case, a CORBA::Any type is used).

The notification service adds to the event service support for event filtering and for QoS specification and enforcement. Filters can be set at different locations along the communication path. QoS properties, such as reliability, priority and timeout, can be specified in order to influence the way in which channels deliver events. The notification service also introduces support for structured events, which have a well-known format and are able to encapsulate most user-defined event formats.

3. SALE and Virtual Enterprise Applications

As it has been previously stated, in the present context where companies seek to increase their levels of competitiveness, the electronic business has appeared as one of the best ways to be followed. While the most visible part is constantly associated with B2C (Business-to-Consumer), it is practically insignificant compared with the form B2B (Business-to-Business), in which the companies carry out all kinds of transactions among themselves (buying/selling, execution monitoring, cooperative planning of purchases, cooperative product development, etc.) and where the issues of reliability, security and real-time are crucial. It is important to notice that in the B2B scenario the number of simultaneous transactions is usually small compared with the B2C scenario.

In the B2B context, various kinds of organization may exist, generically called Virtual Organizations (VOs). Basically, a VO reflects a group of companies that are available to cooperate through the Internet with the aim of taking advantage of business opportunities. Consequently, an infrastructure of communications and services is necessary for allowing cooperation between companies. Depending on the kind of enterprises and business, different forms of VO are used within this concept, namely those of Virtual Enterprises (VEs), Extended Enterprises and Supply Chains. There is an implicit notion in these organizations of the need for sharing resources and information in a coordinated manner so that the whole sequence of transactions involved in the business is not "merely" carried out, but with the quality that all parties have agreed on. In this context, according to what has been previously mentioned, quality has many different facets, depending on the level being considered.

The SALE platform meets these requirements successfully, focusing mainly on the scenario of VEs – the broadest and most complex – offering a package of services with an infrastructure to allow an adequate execution of services at the application level.

3.1. The Application Framework

The application being developed – known as *SC*^{2} *System* – is based on a multi-agent system(2) for decision-making support that aims to assist the managers of virtual enterprises in monitoring the execution of business processes [15]. The *SC*^{2} system comprises results of three recent international cooperative projects in which our research group was involved: Holos-Massyve

(2) In this context, a multi-agent system is an aggregation of computational processes ("agents") which, based on varied degrees of autonomy, interact among themselves in order to reach a common goal.

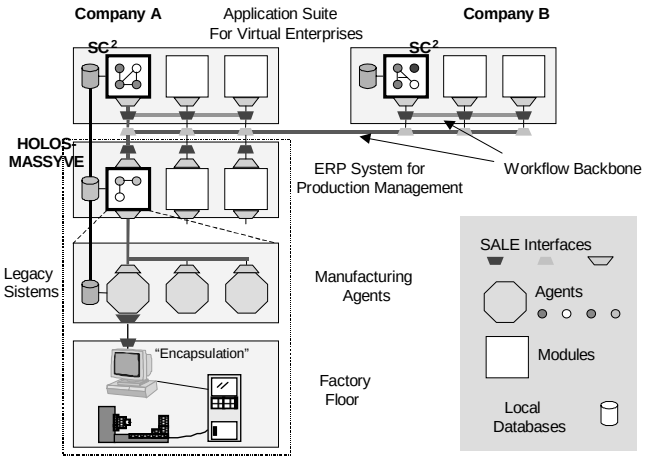

Figure 5. SALE and SC^2 Framework

(http://www.gsigma-grucon.ufsc.br/english/projects.htm#Massyve), ProdNet-II (http://www.uninova.pt/~prodnet), and Damascos (http://bart.inescn.pt/~damascos).

The SALE architecture is seen here as a generic infrastructure for communication services in which applications, the SC^2 system in this case, can be built upon. Figure 5 illustrates the framework composed by SALE and SC^2. In the case illustrated by Figure 5, an hypothetical virtual enterprise is represented, being composed by two companies: "A" and "B". The coordination of this virtual enterprise takes place through instances of *SC*^{2} in both companies. The virtual representation of a company is composed of a set of modules with high-level services. SC^2 is one of these modules. All the intra- and inter-suite communication is done through a workflow system, with information in XML being exchanged through the backbone.

For example, whenever an SC^2 enterprise agent wants some piece of information, it previously subscribes to a list (by "subject") of the workflow backbone; and whenever an agent has the information item desired, it publishes this information on the backbone, and the workflow mechanism sends it on to the agents of the clients on the list. Figure 6 shows the IDL of the proxy for the workflow mechanism supplied by SC^2, which is built upon the CORBA notification service.

```
module WorkFlowBackbone {
  // Proxy for the Workflow Backbone
  interface PublicProxy {
    void   push (in string msg);
    string pull (out string id);
    void   reply (in string id,
                  in string msg);
    ...
  };
  ...
};
```

Figure 6. IDL of the Workflow Backbone used by the SC^2 Agents

In very general terms, it could be stated that the platform *SALE + SC*^{2} serves as a kind of "plug-in" used by a company to perform transactions digitally with other companies (that are also "plugged in"). The platforms for interoperation in a virtual enterprise must encapsulate the legacy systems (ERP, etc.) with which the companies already work. In SC^2, the legacy systems are encapsulated through SALE in the form of CORBA objects (agents, in this case). However, there is no way of avoiding to perform some kind of "re-engineering" in the legacy systems [16]. For example, if company B wishes to monitor the processing of a request sent to component supplier company A, several different legacy systems are necessary for accessing the information from the factory floor, updating the database, encapsulating the desired content into a suitable format, encrypting the data, and sending it through the Internet. Making these services available on the Internet through CORBA objects consists in the necessary re-engineering process.

The Holos-Massyve multi-agent system [17] coordinates the production of a particular factory floor in a component company (scheduling system in a production cell). From the viewpoint of the B2B context, this system acts as a mediator between the needs of the virtual enterprise and the factory floor capacity of component companies. The Holos-Massyve system was conceived to react when there are unforeseen failures of industrial equipment and unpredictable demand. For this purpose, it supports a negotiation between its agents ("manufacturing agents" in Figure 5) seeking to create dynamically the most suitable composition of equipment for a given production plan. Due to the need for permanently monitoring the state of the factory floor, the Holos-Massyve system requires a direct communication with equipments under real-time constraints. All these inter-agent communications in SALE are performed following the traditional client/server model, using the Real-Time Invocation Model illustrated in Figure 4.

The interfaces of the manufacturing agents depend on the kind of activity executed by the legacy systems encapsulated by these agents. Figure 7 shows an example of IDL interface of the agent of a generic controller.

3.2. System Requirements

Based on the description in the preceding section, each level of the hierarchy within an enterprise has different QoS requirements. In this way, for instance, SC^2 agents that represent component companies, dialogue with their peers assuming real-time constraints concerning the workflow mechanism. In the agents at the bottom of the hierarchy, those linked to controllers at the factory floor, the real-time constraints are applied using the Real-Time Invocation Model (Figure 4) on client-server interactions as emphasized in the RT-CORBA specifications.

```
interface Controller {
    void load_program (in short machine,
                       in string filename);
    void start_program (in short machine);
    void stop_program (in short machine);
    void get_status (in short machine,
                     out short status);
};
```

Figure 7. Example of IDL of a Holos-Massyve Agent

```
// Defines QoS parameters of Notif. Service
#include "Notification.QSL";

// Proxy Object uses the notification service
QoS PublicProxy: Notification {
  // Events delivered with best-effort
  reliability = BEST_EFFORT;
  // Priority for delivery of events
  priority = 50;
  // Timeout for pushing events = 2 hours
  push::timeout = 7200;
  // Timeout for pulling events = 2 hours
  pull::timeout = 7200;
  // Timeout for replying events = 1 hour
  reply::timeout = 3600;
};
```

Figure 8. QSL of the Workflow Backbone

QoS of the Workflow Mechanism used by the SC^2 Agents. For the SC^2 agents, that is, at the management level of VEs, the exchange of events through the workflow backbone is performed by the notification service.

The delivery of events follows the best-effort policy, since time requirements at this level are less strict. The QoS requirements specified in the notification service and used by the workflow backbone are described, taking the following parameters as a basis:

- Reliability: guarantee of event delivery (best-effort or persistent);
- Priority: priority of event delivery;
- Timeout: expiration time of an event, after which it can be discarded.

The values of QoS parameters are obtained statically from the QSL specification, shown in Figure 8, or at run-time when these values are altered through the IDL interface of the QoS object, shown in Figure 3.

In the sending of events, timeouts are typically of a few hours, due to the nature of the activities executed at the SC^2 level. Values of QoS parameters are inserted by SALE in the body of the event by the interceptor (see Figure 2). The event is then transmitted by the notification service, taking into account the required timeout.

QoS of the Manufacturing Agents. In the case of Holos-Massyve agents, the real-time constraints are observed by the platform during the execution of method

calls by using the Real-Time Invocation Model. The QoS requirements for manufacturing agents are described based on the following parameters:

- Timeout: the maximum time the client waits for the return of a call;
- Deadline: the maximum time for the execution of the method on the server;
- Priority: priority of execution of the method on the server.

The values of QoS parameters are obtained from the QSL specification (Figure 9), and can be altered at run-time through the QoS object provided by SALE (Fig. 3).

The calls to the manufacturing agents are performed using the programming model previously described in Figure 4, using the corresponding values of timeout and deadline. The deadline is calculated in terms of the defined timeout, considering the estimated time spent for the client processing and transmitting the input and output data. At the moment of the call, the value calculated for the deadline is obtained by the interceptor through the QoS object on the client side, inserted into the context of the call, and sent in the request. The deadline corresponds to a time interval that will be transformed into the time value using the server clock, which eliminates the need of synchronizing clocks or of GPS receivers. The request is intercepted on the server side and transferred to the real-time scheduling service (Figure 4). The priority value is generated in terms of the deadline of the call and of the real-time scheduling policy (in this case, EDF [18]). This value is inserted into the request in the field corresponding to CORBA priority, and used by the POA (Portable Object Adapter) to define the priority of the dispatch thread used for executing the method at the server.

4. Results and Related Work

Based on the results previously obtained by using the SALE platform in supporting the coordination of virtual enterprises and in the coordination of manufacturing cells, the suitability of SALE and its extensions of RT-CORBA to the scenario of the application described above could be affirmed beforehand.

The Real-Time Invocation Model presented in Figure 4, does not define a deterministic communication protocol and uses best-effort policies for real-time scheduling. Although this best-effort model does not guarantee the enforcement of all real-time constraints, its use for agent communication at the lower levels does not represent a problem for the application in question. In discrete manufacturing, where the operations (drilling, assembling, packing, etc.) have beginning and ending dates planned within a given standard execution time, deviations of minutes commonly occur in a work shift.

```
// Defines QoS parameters of RT-CORBA
#include "RT-CORBA.QSL";

// Object Controller uses RT-CORBA
QoS Controller: RT-CORBA {
  //Estimated Communication Time = 20ms
  const float COMM_TIME = 0.020
  // Timeout to start the operation = 60ms
  start_program::timeout = 0.060;
  // Timeout to stop the operation = 120ms
  stop_program::timeout = 0.120;
  // Deadline calculated based on the timeout
  if (timeout > COMM_TIME)
    deadline = timeout – COMM_TIME;
  else deadline = 0;
  // Priority calculated based on the deadline
  if ((deadline*1000) < (MAX_PRIO-MIN_PRIO))
    priority = MAX_PRIO – (deadline * 1000);
  else priority = MIN_PRIO;
};
```

Figure 9. QSL of a Holos-Massyve Agent

This means that best-effort and soft real-time solutions are acceptable in this application context. When these applications require more critical constraints on real-time communication, these requirements can be guaranteed by legacy systems (e.g., instrumentation networks).

In continuous manufacturing, due to its greater complexity, these solutions are not acceptable. Since the processes involved usually have critical tasks, such as pressure adjustments, opening/closing of valves, precise readings of volumes, etc., the perfect control of response times is nearly impossible using best-effort solutions. In this scenario, a difference of tenths of a second in the response time, either more or less, can incur great damages in the production process.

The evolution of supports for specifying and obtaining services with QoS guarantees has been based mainly on providing mechanisms for communication services with real-time constraints. However, some supports have shown QoS requirements related to reliability and security.

On the QoS specification level, we point out the work of the HP Labs [19] involving real-time, reliability and security requirements. Their work is based on language constructions that define quality requirements to be observed statically in services, but it does not define mechanisms for obtaining QoS requirements. The OMG itself is considering the possibility of standardizing ORB interfaces for QoS specification.

Other projects, such as QuO (Quality of Service for CORBA Objects) [20], intend to provide applications built on CORBA with QoS specification and enforcement mechanisms. QuO provides a set of QoS specification languages, allowing static definition of QoS constraints, and adds QoS enforcement mechanisms to the ORB. We believe that SALE provides more flexibility to the user, since it allows static and dynamic specification of QoS requirements, through QSL and the QoS Object

respectively. Due to the use of mapping scripts and application-level QoS mechanisms (i.e. the QoS Object), SALE can learn to interpret and enforce user-defined QoS parameters, while the ORB-based mechanisms provided by QuO would have to be modified to do so.

The issues of fault tolerance and security are not yet considered in current implementations of the SALE architecture. With the evolution of SALE and SC^2, these issues will be integrated to the support.

5. Conclusions

The SALE project adds support for QoS to the CORBA environment, providing mechanisms for specifying and enforcing the real-time, fault-tolerance and security constraints of applications.

In this paper we have shown how Virtual Enterprises can use SALE in order to enforce real-time constraints. Two communication mechanisms, the workflow and the Real-Time Invocation Model, both defined according to the CORBA specifications, were employed at different levels of the application hierarchy. These mechanisms are vehicles for implementing different QoS constraints present in the application context of virtual enterprises.

References

[1] J. Browne, P. Sackett, J. Wortmann. Future Manufacturing Systems: Towards the Extended Enterprise. Computer in Industry, Vol. 25(3), 1995.

[2] Object Management Group. The Common Object Request Broker: Architecture and Specification. Revision 2.6.1. OMG Document formal/02-05-08, May 2002.

[3] Object Management Group. Fault-Tolerant CORBA: Joint Revised Submission. OMG Document orbos/99-12-08, December 1999.

[4] Object Management Group. Security Service: version 1.7. OMG Document formal/01-03-08, March 2001.

[5] Object Managemet Group. Realtime CORBA 1.0: Revised Submission. OMG Document orbos/98-12-10, December 1998.

[6] Object Management Group. Notification Service Specification. OMG Document formal/00-06-20, June 2000.

[7] L.C. Lung, J.S. Fraga, J.M. Farines, M. Ogg, A. Ricciardi. CosNamingFT: A Fault-Tolerant CORBA Naming Service. Proceedings of the 18th IEEE International Symposium on Reliable Distributed Systems. Lausanne, Switzerland, October 1999.

[8] L. C. Lung, J. S. Fraga, J.-M. Farines, J. R. Oliveira. Experiências com Comunicação de Grupo nas Especificações Fault Tolerant CORBA**.** Proceedings of the 18th Brazilian Symposyum on Computer Networks, Belo Horizonte, Brazil, May 2000.

[9] M.S. Wangham, L.C. Lung, C.M. Westphall, J.S. Fraga. Integrating SSL to the JaCoWeb Security Framework: Project and Implementation. Proceedings of the 7th International Symposium on Integrated Network Management – IM'2001. Seattle, USA, May 2001.

[10] C. M. Westphall, J. S. Fraga. Authorization Schemes for Large-Scale Systems based on Java, CORBA and Web Security Models. Proceedings of the IEEE International Conference on Networks – ICON'99, Brisbane, Australia, Sept. 1999.

[11] F. Siqueira. Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL. Proceedings of the 20th Brazilian Symposium on Computer Networks, Búzios-RJ, Brazil, May 2002.

[12] C. Montez, J. Fraga, R. S. Oliviera, J-M. Farines. An Adaptive Scheduling Approach in RT-CORBA. ISORC '99: IEEE International Symposium on Object-oriented Real-time distributed Computing. Saint-Malo, France, May 1999.

[13] J.S. Fraga, J.-M. Farines, C. Montez. Um Serviço de Tempo Global para Sistemas Distribuídos de Larga Escala. XVI Brazilian Symposium on Computer Networks, Rio de Janeiro, Brazil, May 1998.

[14] D. C. Schmidt et al. TAO: A High Performance Endsystem Architecture for Real-Time CORBA**.** IEEE Communications Magazine, 14(2), 1997.

[15] R. J. Rabelo, R. V. Vallejos. A Semi-Automated Brokerage for a Virtual Organization of Mould and Die Industries in Brazil. First IFPI Conference on E-Commerce, E-Business, E-Government, Zurique, Switzerland, October 2001.

[16] L. M. Camarinha-Matos, H. Afsarmanesh. Further Developments in Virtual Enterprises. In: Infrastructures for Virtual Enterprises, Kluwer Academic Publishers, October 1999.

[17] R. J. Rabelo. Interoperating Standards in Multiagent Manufacturing Scheduling Systems. International Journal of Computer Applications in Technology (IJCAT), 2000.

[18] C.L. Liu, J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20(1), 1973.

[19] S. Frolund, J. Koistinen. Quality of Service Aware Distributed Object Systems. White Paper, Hewlett-Packard, 1998.

[20] D. Waddington, G. Coulson and D. Hutchinson. Specifying QoS for Multimedia Communications within a Distributed Programming Environment. Lecture Notes in Computer Science, Vol. 1185, Barcelona, Spain, November 1996.