

UNIVERSITEIT VAN AMSTERDAM



DATA COMMUNICATION INFRASTRUCTURE FOR DISTRIBUTED REAL-TIME SIMULATION

MARCO J. PATRICK

0272396

Master of Science Thesis
Department of Computational Science
Universiteit van Amsterdam
Amsterdam, The Netherlands

August 2004

Summary

Computer simulation is commonly used to analyse the behavior of physical systems. When the simulated behavior is time-dependent and the simulated system consists of interacting components then the correct co-ordination in time of these simulated components is critical. In other words, the states of the different components must be consistent in time at each moment of interaction. For simulations in which *simulated components* interact with *non-simulated* (real) components, the timing interaction between the simulated and the real components requires a so-called real-time simulation system. Typical examples of such *non-simulated* components are hardware equipment or human interaction. Typical examples of such *simulated* components are computer models of physical motion or of visual appearance (as found for example in flight simulators). In real-time simulations these simulated components are executed under strict control and on dedicated (high-performance) computer systems, as such assuring that the strict requirements on the time-consistency of the simulation are fulfilled. In certain cases, however, it is inevitable or desirable that these simulated components are executed on a more generic, non-dedicated computer system, which in its general form consists of a network of computers. In such cases it is a challenge to meet the real-time simulation time-consistency requirements for time dependent simulation.

Two different kinds of distributed system architectures that enable distributed real-time simulation (i.e. real-time simulation on computer networks) have been investigated in this study, namely the High Level Architecture (HLA) and Real-Time Common Object Request Broker Architecture (RT-CORBA). HLA is known as the latest technology in the area of distributed simulation while RT-CORBA is normally applied in more general distributed systems (such as distributed databases, distributed control systems, etc), not necessarily simulation systems. Both architectures have been investigated extensively from literature, from which it was concluded that RT-CORBA is currently the most viable for simulations with strict real-time requirements. Therefore an implementation of RT-CORBA (TAO: The ACE ORB) was selected and applied in a case study of distributed real-time simulation. In this case study, two real-time simulation environments that are operational at NLR, EuroSim and Tower Research Simulator (TRS), have been coupled in an integrated analysis of real-time simulation of vehicle motion in a virtual airport environment, where the interface and the time scheduling execution of the two was based on the RT-CORBA system. Several integrated real-time simulation scenarios have been successfully evaluated and results were compared with an ad-hoc implementation of the integrated simulation system that was based on Remote Procedure Call (RPC) mechanisms.

CONTENTS

1	INTRODUCTION	1
2	REAL-TIME SYSTEMS	4
2.1	Real-Time Simulation Systems	5
3	MIDDLEWARE FOR DISTRIBUTED REAL-TIME SIMULATION SYSTEM: LITERATURE SURVEY	8
3.1	High Level Architecture (HLA).....	10
3.2	Common Object Request Broker Architecture (CORBA).....	11
3.2.1	Standard CORBA	12
3.2.2	Real-Time CORBA	13
3.3	Time Management Service in HLA and in Real-Time CORBA	16
3.3.1	HLA Time Management Service	16
3.3.2	Increasing System Predictability in RT-CORBA	17
3.4	Distributed Real-Time Simulation Architecture: Conclusion	19
4	DISTRIBUTED REAL-TIME SIMULATION SYSTEM: EXAMPLE	21
4.1	EuroSim	22
4.1.1	Introduction.....	22
4.1.2	Simulation Life Cycle	22
4.1.3	Model Editor	23
4.1.4	Schedule Editor	24
4.1.5	Simulation Controller	24
4.1.6	Test Analyzer	25
4.1.7	EuroSim Daemon	25
4.1.8	Simulation States	26

4.1.9	Event Channel	27
4.2	Tower Research Simulator	28
4.2.1	GEAR Simulator Middleware	28
4.2.2	Configuration of the TRS System	29
4.2.3	Simulation states	31
4.2.4	Time Management	32
4.3	RT CORBA Implementations: The ACE ORB (TAO)	33
4.3.1	TAO Real-Time Event Service Configuration	34
4.3.2	TAO Real-Time Scheduling Service Configuration	36
5	CASE STUDY: DISTRIBUTED REAL-TIME SIMULATION OF AIR- PORT VEHICLE MOTION	39
5.1	Introduction	39
5.2	Vehicle Model	40
5.3	The simulation scenario	41
5.4	Distributed Simulator System Implementation	42
5.5	Distributed Real-Time Simulation System Using TAO	44
5.6	Results	45
5.6.1	Data Transfer Verification	46
5.6.2	The Latency of Time Data Transfer	47
6	DISCUSSION & CONCLUSIONS	50
6.1	Discussion	51
6.2	Conclusions	51
6.3	Recommendations	52
	APPENDIX	57
A	High Level Architecture Components	57
A.1	HLA Interface Specification	57

A.2	HLA Object Model Template	58
A.3	HLA Rules	58
B	CORBA Components	59
B.1	ORB Core.....	59
B.2	OMG Interface Definition Language (OMG IDL)	59
B.3	Language Mappings.....	59
B.4	Servant	60
B.5	Portable Object Adapter (POA).....	60
B.6	Dynamic Invocation Interface (DII)	60
B.7	Dynamic Skeleton Interface (DSI)	60
C	TAO ORB Configuration	61

1 INTRODUCTION

Computer simulation is widely used in the process of design and analysis of physical systems. It is a way to predict and understand the behavior of the system in virtually any level of detail.

For example in the design of whole aircraft or aircraft level systems (e.g. flight control systems), it is prohibitively expensive in terms of budget and time to build the complete aircraft physically and analyze the behavior of it. Such costs can be largely avoided by using computer simulation. Moreover one may gradually increase the detail of the simulation since each component of an aircraft can be represented as an element in the computer model. A change in a design requires the designer only to change the computer program or adjust some parameters and re-run the simulation to see the effect. In this sense, computer simulation will speed-up the design process and in addition, it allows for easy storage and re-use of model components.

Computer simulation is often applied to time-critical processes or events. Some examples are flight simulation, evaluation of scheduling scenarios of airport traffic, scheduling of instruments in nuclear power plants, etc. In all cases, the system shall be represented as a computer model, executed and analyzed in a computer simulation according to certain time constraints.

For simulating such time critical events, there are two important time references that are used. The first one is *simulation time*, defined as an abstraction used in the simulation to model time in the physical system that is simulated. *Wallclock time* as the second reference, is defined as the time (according to the standardized Greenwich Mean Time (GMT) [9]) elapsed during the execution of the computer simulation. The simulation time can be compared to the current value of wallclock time by reading the hardware clock that is maintained by the operating system. Whenever the simulation time is paced exactly as the wallclock time, the simulation is referred to as being executed in "hard real-time". If the simulation is allowed to have some delay (i.e. the simulation time lags behind the wallclock time) then this is called "soft real-time" simulation.

Time critical simulations are typically used if there are hardware equipments or human interactions involved inside the simulation loop, normally referred to as *hardware-in the-loop* (HWIL) and *man-in-the-loop* (MITL) simulations [33]. Some examples of MITL are pilots flying in flight simulators and air-traffic controllers controlling airport scenario simulations. MITL gives direct assessment of the realistic aspect of the simulation. For example in a flight simulator, the input from the steering device has to be synchronous with the movement of the aircraft, otherwise the pilot will feel the unrealistic aspect of the simulation. HWIL simulation is achieved by putting real hardware or physical devices in the simulation loop. The real hardware/device in this case may be considered as a "perfect model" that is part of the simulation system, and that allows no delay or latency at all for the

real-time simulation (i.e. hard real-time).

The implementation of a real-time simulation system can be done in a single machine with single processor or multiple processors (shared memory). In either case, the scheduling of the execution for each component in the simulation system will strongly depend on the operating system (OS) of that particular machine. In addition, dedicated software is available that supports the user in the proper timing, scheduling and execution of the simulation components. An example of such a simulation environment for real-time simulation is EuroSim [34]. In general, when using such a real-time simulator for a single machine, each simulation component is defined as a *model* which can interact with other components. For giving the real-time execution guarantee, a schedule is embedded to each model when the interconnection among them is defined. Having that schedule, the simulator will depend on the OS to arrange the execution such that the real-time execution can be achieved.

Having a real-time simulation system running in a single machine is not always feasible. For example if different models shall be integrated in a single simulation system but each model has been developed in a specific environment (using different software, OS, processor architecture, transport protocol, etc). Similar problems will arise if legacy simulation models are re-used. For the simulation system involving HWIL, the hardware itself is in most cases not portable. Hence there is a need for a generic and common interface where different models can be easily interconnected into a distributed simulation system while the real-time execution is still guaranteed.

All the models in an integrated distributed simulation system are considered as black-box components that provide only their input and output with possibly different interfaces. A coordinator is needed to handle the diversity in the model's interfaces, including the diversities in data representation. In such a case, output(s) of one model must be suitable as an input for other models. Performing the execution of simulation in real-time requires that the coordinator has to be equipped with the functionality of a scheduler as well, since the data exchange among the models has to be performed in such a way that all the deadlines are met. Having a distributed simulation system that is based on a standard and that can handle the diversities in data representation while maintaining the real-time performance is a major target in this study. The developments in the area of distributed computing have produced several distributed-systems, known as *middleware*, that contain such functionality. Some middlewares are designed for a specific simulation system while others are designed to deal with a more general application, not limited to real-time simulation. In this study, the assessment focused on two types of middlewares (HLA and RT-CORBA) that are commonly used in distributed real-time simulation. One of the most important aspects investigated is the capability of the middleware to provide services which preserve and guarantee the real-time execution of the distributed real-time simulation system.

This report will first introduce the distributed real-time system followed by a survey of middleware implementations that were investigated in this study. Chapter 5 presents a case study where a system consisting of several real-time simulators will

be inter-connected via the middleware. A distributed real-time simulation system with RT-CORBA as a middleware to connect and integrate two types of real-time simulation environments has been implemented and tested.

2 REAL-TIME SYSTEMS

A Real-Time System is defined as a system in which its correctness depends not only on the logical result of the computation, but also on the time at which the results are produced [35]. Real-Time does not necessarily mean "real fast"; the emphasis is on the time when the computational deadline has to be met. Deadline in this case is defined as the instant of time by which its execution is required to be completed [20].

The most important point in a real-time system is that the time when the system can provide the computational result has to be *predictable*. In real-time system, the guarantee to meet computational deadline is an important aspect. This guarantee can only be provided if the computational time needed by the system is predictable.

A *Hard real-time* system has a stringent requirement that all of its components have to meet all the deadlines. A single missing deadline means the failure of the whole system. In other words the computation is considered successful only if the correct result (of each of the system's components) is delivered exactly at the required time constraint. To illustrate this in some more detail, figure 2.1 shows the quality of the computation versus the time advance to show the hard-real time requirement. The higher the quality of the computation, the more contribution the computation gives to the system objectives. It means that if the result of the computation can be provided within the period between "start-time" and "deadline", this result is considered as useful. When given outside this period, the result is considered as useless and it means a failure to the system. This failure is represented in the figure as the "quality of computation" being equal to zero outside the allowable period. An example of a hard real-time system is a flight simulator system. Among others, the motion of the aircraft should be realistically, accurately and timely be computed and translated to changes in the visual system that represents the simulated pilot's view. Similarly, when considering the hardware in the flight simulator system, this HWIL can be considered as a simulation component which requires hard real-time guarantee while exchanging data with other simulation components. For example if that HWIL requires data at the frequency of 40 Hz, providing the data at the rates of 39.9 Hz will result in a failure of the simulation system.

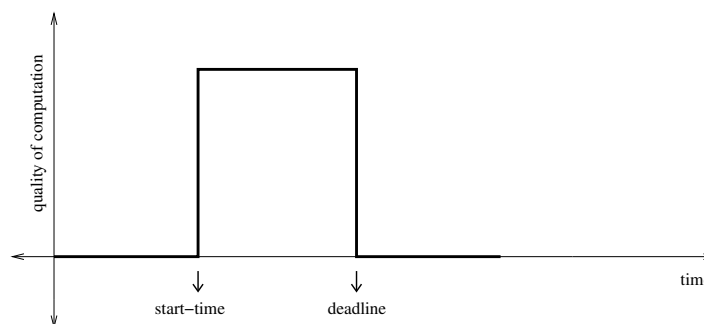


Figure 2.1. Illustration of a Hard Real-Time System

The other type of system, known as *Soft real-time* system, relaxes the requirements a little bit by allowing several missing deadlines. The value of the computation is considered as still being useful as far as the amount of "meeting the deadlines" are still below an acceptable threshold. Figure 2.2 illustrates the soft real-time system. Compared to figure 2.1, the "quality of computation" is not suddenly zero when the result is provided after the deadline has passed. The usefulness of the result, expressed as the quality of computation, starts to decrease as the time advances after the deadline. The rate of decrement of the "value of computation" after passing the deadline depends on the criteria of the system.

A pilot steering an aircraft model in the a flight simulation is an example of such a system. Whenever the model experiences a certain latency for the data that it receives from the steering device, the pilot will notice the unresponsive behavior of the model at the visual system, simulating the pilot's view. Nothing serious happens as long as the latency does not grow or accumulate. The rather relaxed time requirements in soft real-time simulation are typically related to the MITL. As MITL mostly acts as an observer in the simulation system, even if he/she notice the delay in the simulation display, he/she can still tolerate it and let the simulation system keep on going.

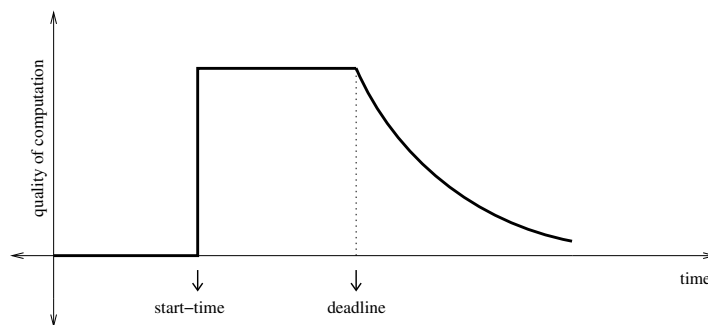


Figure 2.2. Illustration of a Soft Real-Time System

2.1 Real-Time Simulation Systems

One example of real-time system application is in the area of simulation, where the system is called real-time simulation system. Since it simulates the physical system, there are different reference of time that are used. There are two important notions of time in the real-time simulation system, namely:

- *Wallclock time*, is the physical wallclock time which elapsed during the execution of simulation program (for example, 10:00 to 10:15 AM on July 15, 2004)
- *Simulation time*, is the elapsed "virtual" simulation time (representing the time inside the simulation system) which occur during the execution of sim-

ulation program. In general it is a range of floating-point number like [0.0 7.5]

Figure 2.3 illustrates the wallclock time - simulation time plane. It illustrates the real-time requirement in real-time simulation systems, concerning the time-deadline, t_d which is defined as the latest time at which all the simulation's components have to provide their result to the system.

Hard real-time simulation system is a system of which all of its components can provide the simulation result to the system at wallclock time no later than the point h in figure 2.3. This also holds in case all the results can be provided below the point h . In other words, the real-time simulation system is achieved whenever $t_{wc} < t_d$.

For other system's requirements, which is referred to as *soft real-time simulation*, the simulation result may be delivered with some delay (at an arbitrary size) after the deadline t_d , namely at the time $t_{wc} < t_d + \Delta t$. The value of Δt depends on the requirement imposed by the system. It should be noted that Δt should not accumulate in case of sequential events.

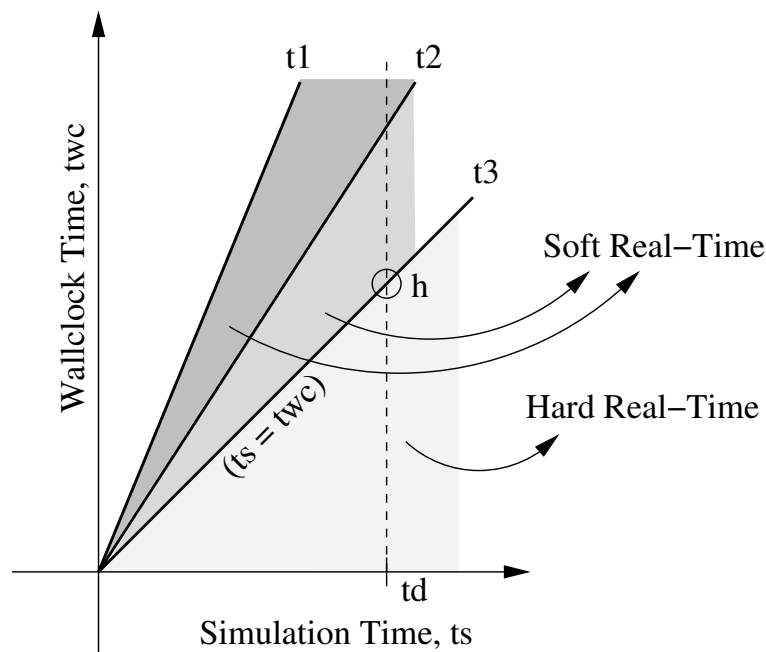


Figure 2.3. Hard and Soft Real-Time in Simulation System

To meet the demand of the system regarding requirement of the "soft" and "hard" deadline, the system behavior has to be predictable so that a strategy can be developed to meet system requirements. One way to increase the predictability of the system is by imposing a requirement in each simulation component in terms of time that is required to provide simulation/computational result. Based on this time requirements that is given by each simulation component, a scheduling strategy can be built in such a way that guarantees to meet the deadline can be provided.

In a real-time simulation system consisting of several components where all of them have the requirement to meet deadlines, a scheduler is needed to coordinate the order of execution of each component and the access to each component resources. The scheduler must be able to predict these two elements, otherwise it cannot guarantee that the simulation deadlines of the system will always be met.

The scheduling of a real-time simulation system is related to the management of the system resources of the computer. For a single processor machine, the system resources are the OS threads and the main memory while for a multiple processor machine (e.g. shared-memory type of system), processor is also a system resources which has to be scheduled.

For a real-time computer simulation running in more than one computers, the system resources are the operating system's threads, memory and network bandwidth. At the lowest level of the system, the access management to the resources, namely the threads and the memory mainly depend on the operating system. At the higher layer (i.e. at the transport layer, refers to the OSI model [38]), the use of the network bandwidth are mainly managed by the transport protocol. For hard real-time simulation, ideally both of these resource managers, namely the operating system and the transport protocol, have to guarantee the real-time performance. Having Real-Time Operating System (RTOS) and real-time transport protocol, the source of unpredictable behavior of the system can be reduced.

In meeting the specified time requirements, a scheduling theory is needed to arrange the use of the available computation resources. There are several different types of algorithm in scheduling theory that address the problem in meeting the specified time requirements. Based on the time when the schedule is calculated, there are two types of scheduler, namely *static* and *dynamic* scheduler. The static scheduler composes the schedule based on the offline analysis before runtime. It requires prior knowledge of the feasibility of the schedule but on the other hand it requires little runtime overhead since an optimum schedule is determined during the offline analysis.

There are some real-time systems where its behavior changes dynamically, such that the static scheduler is not appropriate anymore. For compromising with such a dynamic system, a dynamic scheduler determines its schedule during the runtime. It can deal with the unexpected behavior of the system by adjusting the schedule dynamically.

In distributed real-time simulation system, an optimal scheduling strategy that can be used highly depends on the system behavior. In that sense, no single scheduling strategy can perform best in all environments [6]. In order to support and integrate different kind of simulation systems in realizing distributed real-time simulation system, it is necessary that the middleware can also provide different scheduling strategy that user can choose to meet the demand of his/her application.

3 MIDDLEWARE FOR DISTRIBUTED REAL-TIME SIMULATION SYSTEM: LITERATURE SURVEY

One of the goals in designing a distributed simulation system is to increase the performance of the computer simulation effectively while maintaining the behavior of the system. The result of the simulation has to be exactly the same when the execution of the simulation system is moved from a single machine into a distributed machines. For a simulation system with real-time guarantee requirement, this means that the execution of each model meets the deadline when the simulation is executed either in a single or distributed machines.

Before dealing with the real-time requirement, there are already some issues that have to be handled in such a distributed simulation system. The data structure and representation of each machine involved in the simulation might be different due to differences in the machine's architecture. The data structure and the language used for the model implementation might be different as well, for example because the legacy-models that were built possibly using an older machine architecture are used. In spite of all these differences, all the models should be able to exchange data and understand each other while also maintaining the execution in such a way that all deadlines in data exchange for each model will always be met.

For dealing with such diversities in the operating-system and data structures, one can use a *middleware*. It will manage a distributed system in such a way that each different and independent machine can perform a task assigned to it in coherence with other machines. In a distributed simulation system, middleware also has to arrange the coordination of activities among models involved in the simulation system.

Regarding the features of inter-operability, re-usability and real-time performance, there are two main architecture of middleware that are commonly used nowadays. The first option is the High Level Architecture (HLA) [23], which is the most recent effort today in defense simulation community and which has been adopted as standard by United States Defense Modeling and Simulation Office (US-DMSO). Today, HLA is known an IEEE standard 1516 [1].

The HLA has initially been used to integrate many simulators which have been developed so far particularly by US Department of Defense (DoD). It put the emphasis on the re-usability and inter-operability features among those simulators under the concept of federation. HLA is aiming to provide a structure that support re-use of capabilities available in different simulators, which in turn can reduce the development cost of a new system. Some examples of military application using HLA in the area of distributed simulation systems includes:

-
- The Army Digital Leaders Reaction Course (DLRC) to federate the Army Combat Simulation, EAGLE, with Army Command and Control Systems [3].
 - Joint Warfighting Program [45]
 - Interfacing simulations with live systems [22]

Even if HLA originated from military/defense community, it can be used as well for civil/non-military applications. In [2], HLA is used for simulating distributed driving vehicle federation, connecting streetcar federation, barrel-filling station federation and logistic & storekeeping federation. This type of application generalizes the use of HLA not limited to military application.

From the civil or non-military community the most viable effort for the distributed real-time simulation is the Real-Time Common Object Request Broker (RT-CORBA). It is a part of the revised version of standard CORBA, developed by Object Management Group (OMG). In the current version of standard CORBA (CORBA 3.0), the RT-CORBA is included as one of the Specialized CORBA Specifications [26]. RT-CORBA can be used as a scalable middleware for a distributed system in general, not necessarily real-time simulation systems. It put the emphasis on the scheduling service for distributed real-time systems a stringent Quality of Service (QoS) requirement. In such a system, a QoS is define as a timely requirement imposed by the system for executing and delivering the simulation result from each of the simulation model to a simulation system.

Some examples of real-time CORBA applications are the following:

- Manned and unmanned real-time avionics mission computing embedded systems at Boeing/McDonnell Douglas [17], [11], [18].
- Achieving inter-operability and information sharing in network centric battlespace [29].
- Distributed, Real-Time Avionics Domain-Weapons System [4].
- Distributed audio/video streaming service application [14].
- Unmanned Air Vehicle [16]
- and many other United States Army applications [32].

Even before the real-time CORBA specification released, there was an example of CORBA-based project (funded by European Union) called OPERA [41] which built a distributed real-time simulations. It extends the HLA concept to achieve high performance, real-time and flexibility to fullfill the requirements of distributed real-time simulations [41].

Both of these middlewares offer most of the functionality to manage the diversity in the operating system and data structures and to coordinate the data exchange among models that are part of the distributed simulation system. HLA offers an architecture in the form of federation which is managed by the Run-Time Infrastructure while RT-CORBA represents the system as consumer and supplier which are managed by the Object Request Broker (ORB) and supported by several different real-time services. Next section will give a general overview about both architecture.

3.1 High Level Architecture (HLA)

The main goal of HLA is to provide inter-operability and re-usability within a simulation system environment. Each element of the simulation system (not necessarily computer simulator) can be a member of the system and is called *federate*. Two or more federates might have inter-connection to each other and form a *federation*. An HLA system consists of several federations. Federations can have bi-directional communication and work together via *Run-Time Infrastructure* (RTI) of HLA as illustrated in figure [3.1]. The role of RTI here can be viewed as distributed operating system by providing facilities for federates to interact with each other. However, in providing such a functionality for federates, RTI has no knowledge of the semantics of the information passed through it. This is related to the notion of inter-operability which is provided by HLA by keeping the RTI as a general communication infrastructure.

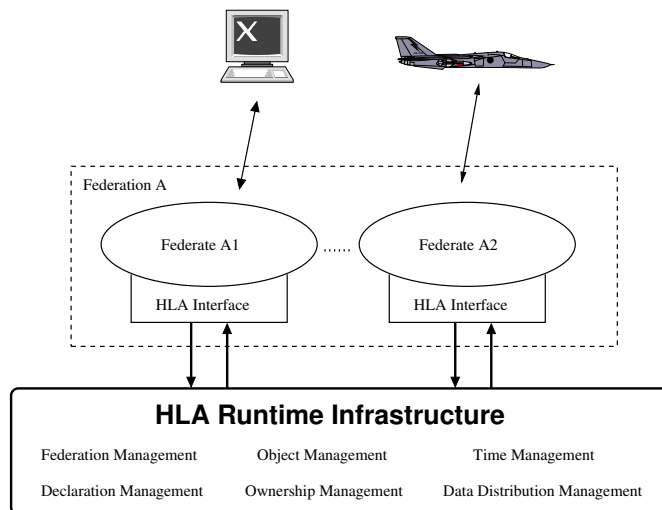


Figure 3.1. High Level Architecture

Each federate represents a simulated entity. In order to specify this entity, HLA applies an Object-Oriented paradigm in the form of object classes. The Object-Oriented notion used here is slightly different compared to the one in software programming. HLA does not specify any method for an object. Instead, a class only specify attributes as a distinct identity. However classes might form a hierarchy with defined attributes inherited from super-classes. All the attributes of an object are

structured and described following the common format, known as the Object Model Template (OMT). OMT dictates the way a federate describes its own property in Simulation Object Model (SOM) and specified the inter-relation of federates within federation in Federation Object Model (FOM). However, OMT does not define the contents of a SOM and FOM. FOM consists of all shared information among the federates which belong to the same federation. Each of the attribute which is defined by the federate has an owner that is responsible to keep an up-to-date information to the RTI. Even if several attributes of one object might have different owner, at any time an attribute is owned only by one owner. This ownership can be transferred to other federate during the execution simulation.

Exchange of information among federates is done by means of subscription. Each federate might subscribe to the specific attribute(s) of other federate(s). Any changes to a certain attribute will be announced by the owner of that attribute to the subscriber via RTI.

HLA also has an other class which is called interaction class. It specifies the relationship between different object classes in terms of event notification, which is not directly related to object's attributes. In a design of federation, either object class or interaction class can be used to model the interconnection among federates as both class are interchangeable. Any federation can be modeled either using object class or interaction class. In general, as the simulated entity has a persistent state, it should be represented as an object. An entity which is characterized by state changes can be modeled as an interaction.

Within HLA architecture there are formally three main defined elements, namely:

- HLA Interface Specification
- HLA Object Models
- HLA Rules

More details about each of these components are explained in appendix A.

3.2 Common Object Request Broker Architecture (CORBA)

In the following section, standard CORBA architecture [24] will be introduced first, followed by the Real-Time CORBA (RT-CORBA) [26] which is now being part of the "Specialized CORBA Specifications" by OMG [25]. RT-CORBA supports all the basic functionalities given by standard CORBA. It extends the CORBA services to support the real-time application by providing a way to schedule the service requests by the clients either statically or dynamically.

3.2.1 Standard CORBA

The main goal of CORBA specification is to provide abstractions for distributed object oriented programming which supports the developers to seamlessly integrate diverse applications into heterogeneous distributed systems. It can be considered as an object-oriented version of the so-called Remote Procedure Call (RPC) [13]. Just like RPC, in general CORBA provides a general interface for object implementation in such a way that the object can always be invoked independently from its location. However CORBA goes much further compared to RPC mechanisms in terms of services that it provides. It automates many common network programming tasks such as object registration, location and activation, request de-multiplexing, framing and error-handling, parameter marshaling and de-marshaling and operation dispatching [42].

Communication in CORBA is done on peer-to-peer basis between "client" and "server". These two roles are determined per-request basis since one component that requests a certain service (client's role) from another component might at the same time act as a provider for other services (server's role) to other components. Figure 3.2 illustrates a simple client server model in CORBA. Client application makes an object invocation to the object implementation at the server side via the ORB.

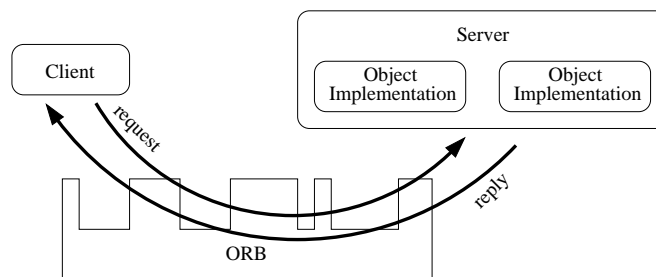


Figure 3.2. A Simple Client Server Model in CORBA

All these components communicate with each other via the Object Request Broker (ORB) which acts as an intermediary. As applications become larger and more distributed, the need for more than one ORB is getting obvious. Consequently, CORBA extends its specification by defining a standard for General Inter-ORB Protocol (GIOP) and its specification for implementation using TCP/IP in Internet Inter-ORB Protocol (IIOP).

The CORBA specification released by the OMG specifies the ORB as the core of the architecture together with several services surrounding the ORB to support the task of ORB to provide a middleware functionality. The most important key-elements that implements CORBA architecture are explained in appendix B.

3.2.2 Real-Time CORBA

A challenging application in distributed simulation is the area of real-time distributed system simulation. Not only the distributed aspect has to be handled, but the guarantee to meet certain deadlines has to be taken care of as well. Data exchange among entities which form a complete distributed system has to be organized and executed in the order such that the result is provided in time, either always before the deadline (for hard real-time system) or below some average for arriving after deadline (for soft real-time system). In such situation, some guarantees in terms of Quality of Service (QoS) has to be given by each element of the system so that the system can create a certain schedule. A QoS in this referring to the timely requirements imposed by the client applications to the server, regarding the time deadline when the server has to send back the simulation result to the client.

As still the case in HLA, there was initially no notion of QoS provided by the CORBA specification. Moreover CORBA is known in the past to exhibit substantial priority inversion and non-determinism [31]. It makes the conventional CORBA unsuitable for distributed real-time simulation. However, as the CORBA specification continuously revised, since CORBA 2.4 there is a new feature which is called Real-Time CORBA that gives a way to specify and guarantee end-to-end predictability for a distributed real-time system. The latest CORBA specification to date is CORBA 3.0 where the Real-Time CORBA (RT-CORBA) is separated from the core of specification and it becomes a specialized CORBA specification [26]. The goal of the RT-CORBA specification is to address the shortcomings of CORBA for distributed real-time systems without sacrificing the main feature of CORBA and without placing a burden on developers of non-real-time system.

For achieving the goal to improve application predictability, RT-CORBA provides a way to bound the priority inversions and manage the system resources end-to-end. They are implemented in the form of thread pools creation (which assign priority to each thread either statically or dynamically) and defining the model for scheduling policies (the options are 'Client Propagated' policy model or 'Server Defined' policy model). In general, the management of system-resources by RT-CORBA are applied to:

- Processor resources: creating a thread pool, priority model and portable priorities.
 - Mapping heterogeneous native OS thread priorities into CORBA global priority
 - Using `SERVER_DECLARED` or `CLIENT_DECLARED` policy which are parts of RT-CORBA priority model policies, QoS can be enforced either by server or client respectively
 - Pre-allocating threading resources from thread pool at the server side to handle client requests efficiently

- Preventing the exhaustion of threads in the threads pool by low priority request by partitioning the thread-pool into subsets, called *lanes*. Each lane has different priority.
- Communication resources: defining protocol policies and explicit binding
 - Use RT-CORBA Protocol Policies to select and/or configure communication protocols. However up to the current version of RT-CORBA specification (RT-CORBA 2.0), only specification for protocol properties of TCP that is available.
- Memory resources: creating a request buffering
 - In case the real-time application need more buffering than is provided by the OS I/O subsystem, client request can be buffered in the ORB.

Basically, in controlling the behavior of the whole system, RT-CORBA provides interfaces to handle threads, priorities and connections which are considered as a system resources. User has two options to have control over all of this system properties. The first option is to arrange scheduling of the invocation of each interface individually such that the collaboration of them act as the real-time system. It will give an opportunity for the system developer to have the finest-grained degree of control over the system. However it also implies that it requires a lot of tuning of each individual interface so that when they are interconnected to each other, the "real-time"-ness of the system is still guaranteed. RT-CORBA specification extends the standard CORBA specification with the addition of several new modules and interfaces to achieve end-to-end predictability and control over the management of resources. Those extention are shows in figure 3.3 with the gray background. For application that does not need real-time CORBA, those gray modules can be omitted.

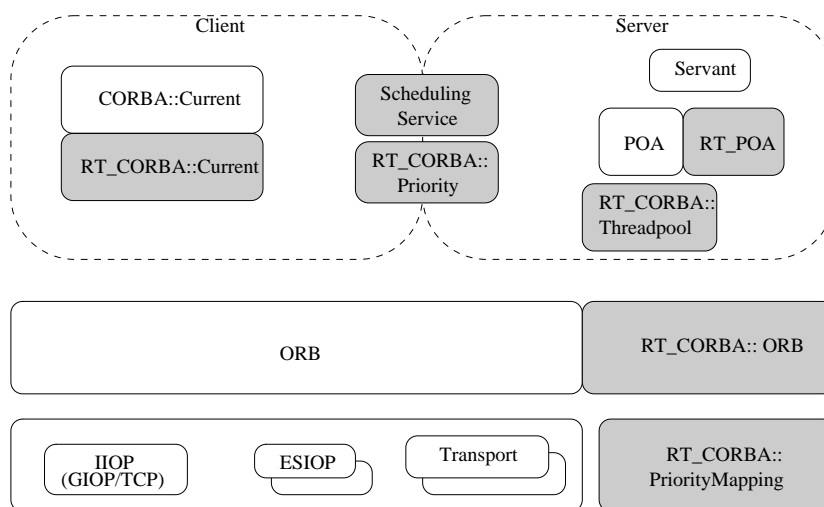


Figure 3.3. Real-Time CORBA Extension

Each of the RT CORBA interface as shown in figure 3.3 extends the functionality of CORBA by:

- `RTCORBA::RTORB`, adds operations to create and destroy other RT CORBA entities, such as mutex, threadpool and real-time policies
- `RTPortableServer::POA`, add operations to allow setting of priority on a per object basis
- Real-Time CORBA Priority, customizing the priority mappings according to a system requirements. The priority model can use either Operating-System (OS) native priority or CORBA priority. Priority mapping defines a mapping between those two
- Threadpools, defining a pool of worker-threads at server-side for handling object invocation. The configuration of this threadpools can be customized to meet the get the best form in relation to system-specific requirements.

Instead of dealing with the individual interface, RT CORBA give a second option, called scheduling service, which provides a higher level of abstraction of the system resources scheduling. Rather than setting the priorities at the thread or connection level, it control the system at the level of application function. However RT CORBA does not dictate a certain policy arrangement. It is up to the developer to choose and define their own policies so that it can be adjusted according to a system-specific requirements.

The scheduling service of RT CORBA 1.0 provides the static scheduling of the resource based on the Rate-Monotonic Algorithm [19]. For a dynamic system, the static scheduling is not able to maintain the stability of the system and adapt the changing in configuration of the system. RT CORBA 2.0 extends the scheduling service by adding a feature of Dynamic Scheduling. Several algorithms can be used to handle such a dynamic behavior of the system while preserving the real-time feature. However, using a pure dynamic scheduling might caused the system to behave non-deterministically under heavy loads and harm the real-time requirement because many critical operations missing their deadlines [10]. The priority inversion might be unbounded which cause the delay of an excessive number of critical operations, waiting for the non-critical operation to finish its job. Some RT CORBA implementation use the hybrid scheduling approach to combine the advantage of those two. That approach will be explained in the next section when discussing the RT CORBA implementation, used for this experiment.

Compared to the standard CORBA implementation, Real-Time CORBA in TAO has some subtle differences. The following describes the features implemented in TAO to support real-time performance [11] which are missing in standard CORBA implementations:

- **Guarantee for real-time event dispatching and scheduling.** Standard CORBA does not have any notion of QoS that can be used by the event consumer to specify their execution and scheduling requirements. TAO provides this QoS specification in terms of `RT_INFO` that consumers can specify and passes it to the suppliers.
- **Specification for centralized event filtering and correlation.** It is possible to implement filtering in standard event channel of standard CORBA implementation by embedding additional filtering service to the event channel. However it will increase the overhead as the event has to pass additional element, namely the filtering service. TAO's approach to provide filtering service is by providing filter and correlation mechanisms that allow consumers to specify logical AND and OR to describe event dependencies.
- **Support for periodic processing.** There is a certain requirement for some real-time application that it has to receive a periodic event. Standard CORBA does not allow the customer to specify these temporal execution requirements. TAO provides this service by allowing the consumer to specify event dependency timeout. Consumer can make a request to receive timeout event in case the dependencies are not satisfied within some period of time.

3.3 Time Management Service in HLA and in Real-Time CORBA

Looking particularly at distributed real-time simulation system, one main concern is how the simulation advances its simulation time. For a real-time simulation system that simulates virtual environment, it is necessary that the simulation time is paced synchronously with the wallclock time. Otherwise the unrealistic aspect of the simulation will become obvious.

3.3.1 HLA Time Management Service

HLA depend on its Time Management service to make a decision in advancing its simulation time. This way RTI can provide a guarantee that a federate will not receive any time-stamped-order (TSO) message with a time-stamped less than its current logical time.

There are in general two ways for ordering messages in HLA, namely receive-order (RO) and time-stamped-order (TSO). An RO message is directly queued by the RTI once it arrives, while a TSO message will be time-stamped but not eligible to be sent to the receiving federate until the RTI can guarantee that there will be no TSO message for that receiving federate which has a smaller time-stamp.

This mechanism implies that federates can not advance its logical time until RTI explicitly grants it to do so. There are three different ways for federates to request the advancement of their logical time, namely:

- Time Advance Request (TAR): well-suited for federates using time-stepped mechanism
- Next Event Request (NER): a preferred alternative for event driven federates
- Flush Queue Request (FQR): optimistically synchronized federates to request out-of-order delivery of events.

The time-stamp information is used by the RTI to compute the lower bound on time stamp (LBTS) before it can grant the time advance request from the federates. The algorithm for computing LBTS is explained in [9]. However in applying this mechanism in real-time simulation system will be too expensive. The lack of timeliness requirements in HLA is in fact a critical limitation for HLA to be applied in real-time simulation system. In such a system, the amount and predictability of RTI overhead is an important factor. The inability to specify or guarantee message delay is becoming one factor that has precluded an acceptance of HLA in the simulations and testbeds involving HWIL [40].

As HLA does not include an execution model for real-time simulation, several extensions to the HLA specifications might be appropriate to address that limitation [40]:

- Extending RTI delivery specification to support QoS requests
- Specifying timeliness requirements for RTI and federation ambassador calls
- Providing real-time synchronization requirements for RTI Service processing
- Specifying real-time simulation execution models

3.3.2 Increasing System Predictability in RT-CORBA

The main goal of RT-CORBA is to improve the predictability of ORB so that it can support distributed real-time applications. Inherently, this extension provides RT-CORBA with extended capabilities to handle concurrency which is a fundamental issue in real-time system.

As explained in section [3.2.2], RT-CORBA deals with concurrency by creating thread-pools and scheduling service [25]. The first one gives the developer the finest-grained degree of control over his/her system at the cost of increased complexity. The second choice offers the developer a higher abstraction of system resources scheduling without dictating the policy arrangement. It is up to the developer to configure the most appropriate policies to their system-specific requirement.

Thread Pools

Multi-threading is one way to distinguish different types of service and it creates the possibility to support thread-preemption in order to prevent unbounded priority inversion. RT-CORBA addresses this multi-threading paradigm by defining a specification for thread pool model to manage threads on server-side of Real-Time ORB.

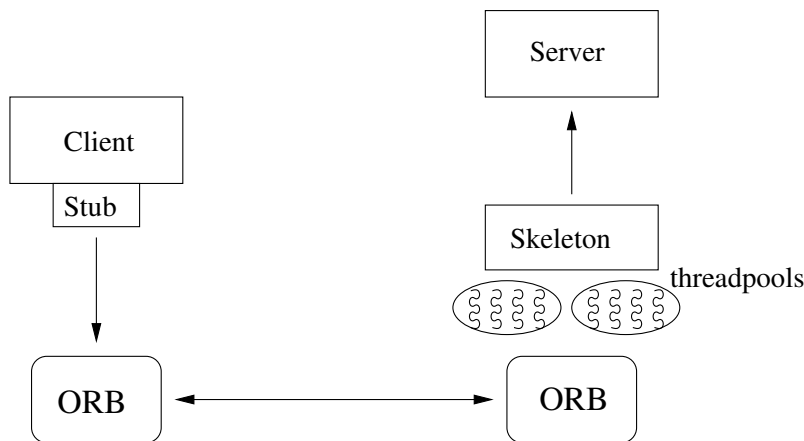


Figure 3.4. Threadpools

Creation of the threadpools offers the possibility for the pre-allocation & partitioning of threads and bounding the threads usage. All of these features help to reduce priority inversion which is the source of unpredictable execution time of certain tasks.

As figure 3.4 shows, a thread pool is a collection of threads that are all separately available to perform work on behalf of the ORB. A typical thread pool consists of two or more threads, all waiting for serving incoming request from client side. Each thread in a pool may be of the same default priority or the maybe grouped together in *lanes*. The left side of figure 3.5 shows a thread pool without lane while the right side show a thread pool with 3 lanes. Each lane has a priority which is inherited to all threads in the same lane. Upon creation, a thread pool may have a number of pre-created static threads and a number of dynamic threads which can be created later if needed. As an example, the left side of figure 3.5 has 12 static pre-created threads and 16 dynamic threads which can be created on demand. The right side of the same figure shows the number of static and dynamic threads for each lane.

RT-CORBA allows a lane with higher priority to borrow a thread from a lane of lower priority in the same thread pool. It should be note that the creation of thread might increase the system performance while at the same time requiring quite a lot system resources, namely memory. In that sense, there is a trade-off that should be considered while using this mechanism.

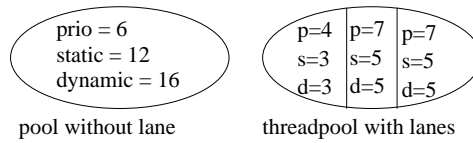


Figure 3.5. Threadpools Without and With Lanes

Scheduling Service

Creating an effective Real-Time Scheduling is relatively complex, especially for a large system size where the scheduling policy has to be enforced uniformly. Scheduling service provides an abstraction of the details of low-level real-time constructs.

Abstraction of the scheduling parameters is done through the use of naming. To define the requirement of application QoS, system designer identifies:

- A Static Set of CORBA activities
- CORBA objects that is used by the activities
- Scheduling parameters in terms of priorities for those activities and objects
- Names that are unique for those activities and objects

Having this information, the scheduling service invokes appropriate RT ORB or RTOS primitives to schedule these activities.

3.4 Distributed Real-Time Simulation Architecture: Conclusion

At the National Aerospace Laboratory (NLR, Amsterdam) where this study is done, there is one project called SmartFED [15], which integrates several existing simulations into a distributed real-time simulation system. However there is no real-time guarantee in this system as it depends fully on the HLA 1.3 RTI-NG implementation that does not have the notion of real-time guarantee.

Realizing that RT-CORBA has never been used before in NLR and it has additional feature compared to HLA which provides real-time guarantee, RT-CORBA is a good alternative to be used as a middleware in distributed real-time simulation. The demand to include hardware in the simulation loop can be met as well while in the case of HLA it is not possible at this moment due to the absence of real-time guarantee notion in HLA specification. Using RT-CORBA, the QoS requirement is conveyed from the client in terms of time-execution requirement which is defined for each client's task which will be executed at the server side. In other words,

client can give a time limit for executing a task at the server side. All of the time requirements in turn will be scheduled by real-time CORBA.

In case of distributed real-time simulation system involving real-time simulator like EuroSim, the real-time performance in each machine executing the simulation model using EuroSim, is already guaranteed. EuroSim provides that real-time guarantee in terms of time execution needed by each model. This timing information in turn can be passed to the scheduling service of RT-CORBA so that the real-time performance in distributed scenario can be preserved.

However the simulation component in such a case is not limited only to simulation environment like EuroSim. Other type of simulation environments can be used as well (more detail is explained in chapter 4). As the number of simulation component involved in a system grows, a thread pool mechanism can used to increase the system performance as explained in section 3.3.2.

4 DISTRIBUTED REAL-TIME SIMULATION SYSTEM: EXAMPLE

To build up a system for demonstrating the use of RT-CORBA, there are several simulation components available that will be considered. In general such a system will be composed from two or more different simulators and the middleware. One example of distributed real-time simulation system will be built as a case study and it consists of several real-time simulation environments interconnected in the distributed scenario.

Nationaal Lucht- en Ruimtevaartlaboratorium (NLR) currently has several state-of-the-art real-time simulators among which two simulators will be chosen. The first one is EuroSim real-time simulation environment that has the main feature to execute simulation models with the hard real-time guarantee. Some simple vehicle model will be used and executed using EuroSim to produce an output of two dimensional orientation of the vehicle. To simulate the distributed simulation scenario, that output will be transferred to the other simulation environment via the middleware.

Tower Research Simulator (TRS) will be used as a second simulation component. In fact it is a real-time simulation environment used for simulating the air traffic controller in an airport. It has not only an advanced feature for arranging airport traffic scenario including the aircraft object orientation and collision detection, but also an advanced three dimensional display environment to show an out-of-window view as seen from the Schiphol Air Traffic Control (ATC). For the purpose of the study, TRS will be used to display the physical model of the vehicle. The vehicle movement in the airport will be dictated by the coordinate, produced by the EuroSim simulation. To demonstrate the general machine architecture that can be used, two EuroSim sessions will be used each controlling a vehicle model. One EuroSim simulation session is executed in linux machine and the other session is executed in SGI-IRIX machine.

As indicated in section 3.4, TAO (RT-CORBA implementation) is considered as the third simulation component, namely as a middleware to manage the distribution of data exchange in the distributed real-time simulation system. One CORBA service and two RT-CORBA services from TAO are important for the purpose of this study, namely Naming Service (CORBA), Event Service (CORBA) and Scheduling Service (CORBA).

In the following section, each element that is used in this scenario will be explained in detail.

4.1 EuroSim

4.1.1 Introduction

EuroSim is a software environment for execution and control of real-time simulation, which supports hard real-time performance when executed on a single machine (single or multi processor) with a real-time operating system (e.g IRIX 6.5 and Redhat linux 8.0 with real-time extensions). It provides an efficient combination of hard real-time capabilities and easy-to-use Graphical User Interfaces (GUI) for importing a simulation model, running and controlling a simulation (possibly with the interconnection to other models) and monitoring & analyzing the result. In arranging the data exchange among the models, EuroSim provides a scheduler which defines the execution order of functions in the model such that the real-time performance is guaranteed. The user or developer of a model specifies the scheduling parameters into the EuroSim Scheduler, which will create the schedule.

EuroSim is a commercial real-time simulation environment, mostly used in the area of space applications, such as F16 fighter aircraft embedded trainer, Simul-taan project, ATV test facilities, simulation of the European Robotic Arm [34]. It supports the use of simulation during the entire project life cycle, from feasibility study to hard real-time operation and training. In the present study, the latest version of EuroSim has been used (mark 1 revision 3/Mk3rev1) which includes the real-time enhancement that supports more hardware interfaces, support for more OS (e.g. real-time linux platform), etc [43].

4.1.2 Simulation Life Cycle

In Eurosim, there are several steps in the design and execution of a simulator, namely:

1. Simulator development: this is an initial step where the models composing the simulation system is created either from scratch or by incorporation of existing models. After the interconnection of these models has been defined, they can be compiled together. The scheduling requirement for each model is also defined in this step.
2. Test preparation: all the initial values, stimuli and output data recording are defined in this step.
3. Test execution: the simulation is run and the result is monitored
4. Test analysis: the recorded result produced during the execution step can be post-processed and analyzed.

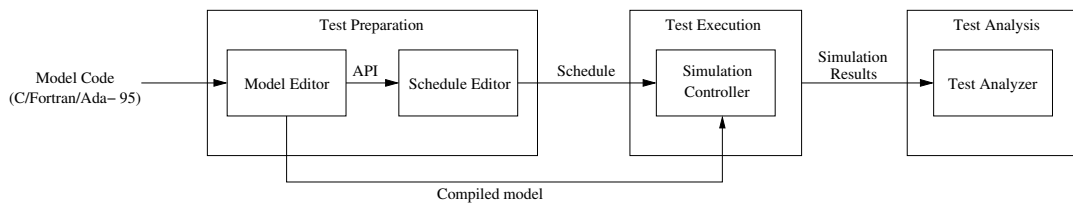


Figure 4.1. EuroSim Simulation Life Cycle

All of these steps are shown in figure 4.1. Within the EuroSim development environment, each block in that figure is provided to the user in the form of a separate tool having a GUI. All the individual tools are described briefly in the following sections.

4.1.3 Model Editor

The *model editor* is used to create generate the interfaces of the models in terms of Application Programming Interface (API). The behavior and characteristic of the model itself are defined using either C, FORTRAN or Ada-95 programming language. It is also possible to specify a model using MATLAB/Simulink ©, Stateflow © or Statemate © and automatically transfer it to the EuroSim environment using the EuroSim pre-process tool called MOSAIC [44]. The API will be generated independent from the programming language so that multiple models can inter-operate seamlessly.

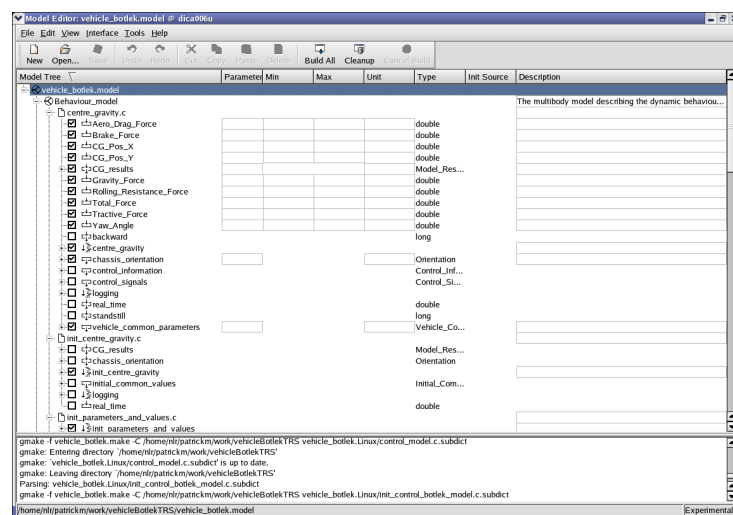


Figure 4.2. EuroSim Model Editor

Figure 4.2 shows a collection of API as a result of the compilation of the simulation model. User can choose which inputs or outputs he/she will use and analyze by selecting the appropriate check-box next to the API.

4.1.4 Schedule Editor

For defining and assessing the feasibility of the schedule, EuroSim has a *Schedule Editor*. Using this tool, the scheduling requirements for each model can be defined based on the API produced by the model editor (the API defines the interface between the model and the EuroSim software). The Schedule Editor will analyze the feasibility of the schedule automatically from the information provided by the user. Schedule editor provides to the user scheduling sequence for four different simulation states, namely "Initializing", "Standby", "Executing" and "Exiting" states.

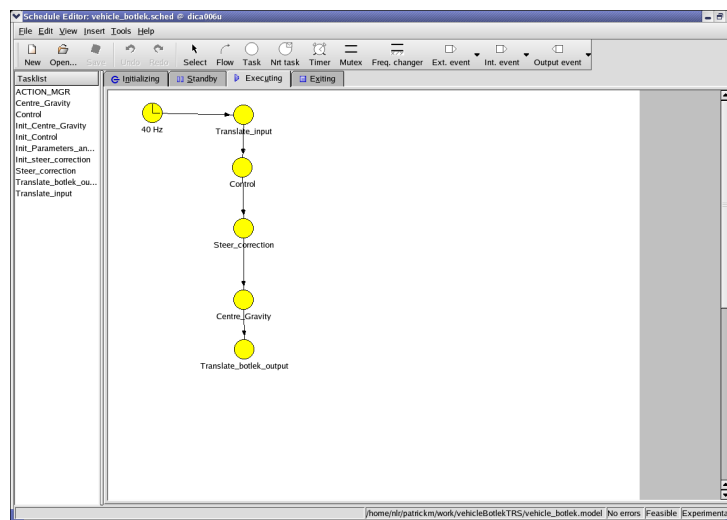


Figure 4.3. EuroSim Schedule Editor

Figure 4.3 shows one particular example of scheduling sequence defined by the user for "Executing" states. It shows several coupled simulation models dictated by the timer of 40 Hz.

4.1.5 Simulation Controller

To control the actual simulation, Eurosim uses its *Simulation Controller* which provides the GUI to control the state of simulation execution. It also can be used to define the initial parameter of the model, preparing the recording of certain values that are necessary for analysis and monitoring.

Figure 4.4 shows the interface provided to the user in the form of GUI. The type of interfaces (whether it is a button, a display-value or a graphical plot) is determined by the user.

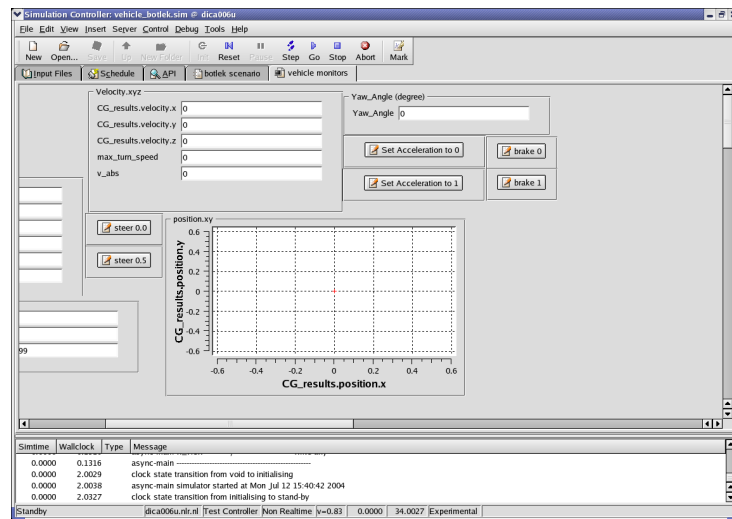


Figure 4.4. EuroSim Simulation Controller

4.1.6 Test Analyzer

After the data has been recorded the user can analyze it using the *Test Analyzer*, which provides a simple way for visualizing the recorded data in the form of graphs. Figure 4.5 shows the plot of a chosen simulation parameter that has been recorded during the execution of the simulation.

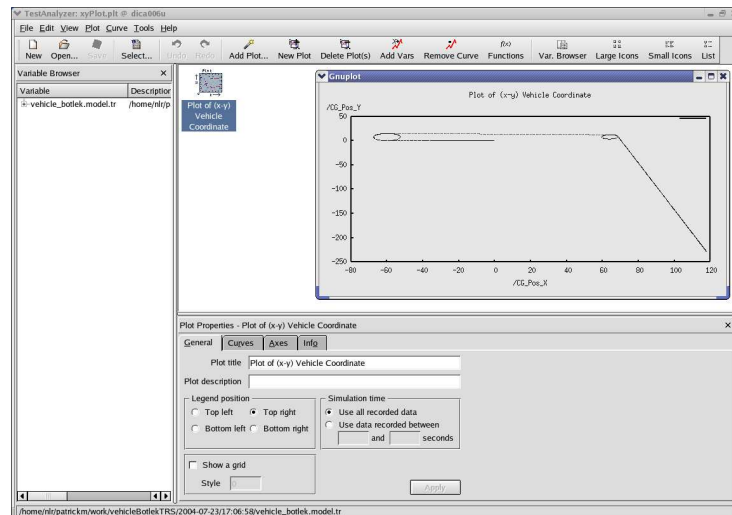


Figure 4.5. EuroSim Test Analyzer

4.1.7 EuroSim Daemon

All of the EuroSim elements described above constitute the EuroSim simulation environment that one can use to develop real-time simulation systems. This environment running on a client machine depends on an underlying server process,

called *EuroSim Daemon* (esimd), which one of its function is to control the permission to start the EuroSim simulator process (in EuroSim, simulation process is called eurosim session). This daemon (esimd) is started automatically when booting the machine on which EuroSim is installed. Each tool in the EuroSim toolset (namely the model editor, the schedule editor, the simulation controller and the test analyzer) will always contact esimd first to acquire the permission to run.

Another function of the esimd is to create the simulation process and control the execution of that process. The esimd is run with root privilege to allow it to set the priority and the scheduling of the EuroSim process so that it is performed with hard real-time guarantee. The state of the running simulation process can only be directly controlled by the esimd. However the user can also control it using the commands that are given by the simulation controller to the esimd via the *Run-time Interface* (RTI) library. This situation is described in figure 4.6 where the simulation controller is shown as a EuroSim client. RTI can establish the connection between simulation controller and esimd using an RPC-based protocol. Once the command is received by the esimd, it passes it to the running simulation process. The esimd also acts as a mediator when the running simulation process sends back the data to the simulation controller for display purposes.

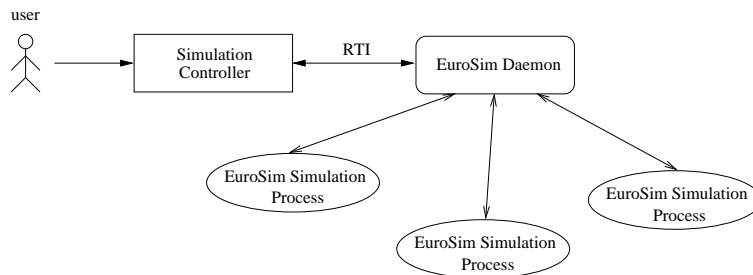


Figure 4.6. The User Controlling the Running Simulation

4.1.8 Simulation States

In EuroSim, a simulation is executed following a certain simulation state. Figure 4.7 shows a state machine containing the available simulation states and state-transitions where each executed simulation started from `rtUnconfigured` state. To change the current state to the next state, a unique event produced by the esimd is necessary to be sent to the running simulation. The esimd will produce that unique event based either on a command received from Simulation Controller or on a command given directly by the user application via EuroSim-RTI. Not only EuroSim tools can communicate with the esimd. Any user-defined application can also communicate with the esimd using EuroSim-RTI library. Whenever a Simulation Controller is used, a user is not aware of the underlying event that is triggered. A GUI button (Init, Reset, Pause, Step, Go, Stop and Abort) is the way provided to the user to change the simulation state. The available simulation states are:

- **rtUnconfigured.** The state after esimd starts the simulation process. It is also the state after the simulation is aborted. The event `eventAbort` is the corresponding event to abort the simulation.
- **rtInitializing.** This is the state of the simulation process after it is initialized and joined one of the event channels. More about the event channel will be explained in the next paragraph. The corresponding event that is needed to reach this state is `eventJoinChannel`.
- **rtStandBy.** The simulation process is ready to be executed at this state. There is no event necessary to be triggered to reach this state since it is automatically entered after the `rtInitializing` state.
- **rtExecuting.** The simulation process executes. The `eventGo` is the corresponding event to get into this state. Once it is started, simulation process can get back to the state `rtStandby` by given an `eventFreeze` event.
- **rtExiting.** The state where the simulation process is stopping (by giving `eventStop` event). This state is necessary when there are some works to do before the simulation process reaches the `rtUnconfigured` state (e.g. leaving the hardware devices in a safe or neutral state).

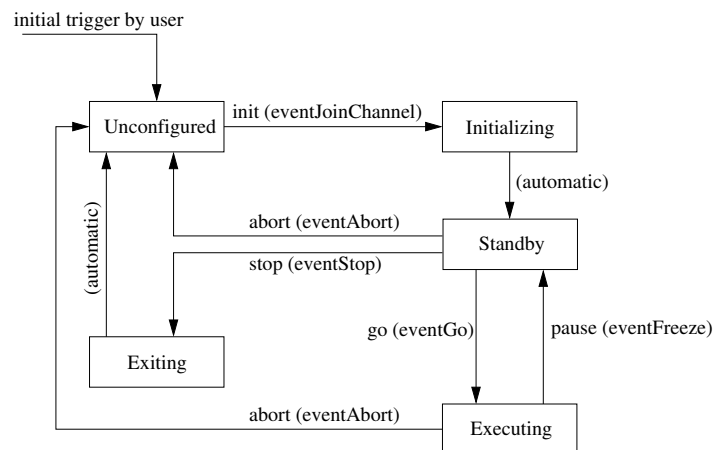


Figure 4.7. EuroSim Simulation States

4.1.9 Event Channel

In order to be able to receive a certain event, a simulation process has to subscribe to one or more event channels. There are four event channels available in EuroSim:

- Control Channel, is used to request and report state changes of the simulator.

- Mission Channel, provides events relating to all activities dealing with the manipulation of scenarios and actions. For a simulator client, it is a mandatory to join this channel.
- Monitor Channel, produce events which monitors the simulation parameters and their values which are interested for the user.
- Schedcontrol Channel, triggering events which are related with the manipulation of the EuroSim scheduler.

4.2 Tower Research Simulator

The Tower Research Simulator (TRS), developed and operational at NLR, is an advanced civil tower Air-Traffic Control (ATC) real-time simulation facility for research, development and validation purposes. It includes a 11×4 meter projection screen showing out-of-the-window view as seen from Schipol ATC tower (or any other tower used in the simulation). The software part of the facility consists of several components (servers) that provide services to each other as defined in their Application Programming Interface (API). The communication among these servers is established based on the middleware, called Generic Environment for Asynchronous Real-time simulation (GEAR) [21], which is a proprietary middleware suite developed by NLR to support distributed real-time simulations (i.e. soft real-time performance).

4.2.1 GEAR Simulator Middleware

Unlike the common client-server architecture, in GEAR there is no fixed role for a component to be either a client or a server. It means that a component which initially has a role as a server can act as a client by requesting a service from other servers. As GEAR is designed for managing distributed real-time simulation, the overhead induced by the client-server communication has to be minimized. GEAR uses a protocol which is called *subscription*.

In such a protocol, a client sends a request for a service but does not wait for the reply. A server which provides that service adds the client into its list of subscribers. Whenever there is an update to a relevant data item, the server will send a reply to a subscribed client. This is particularly useful in a situation where a single request might result in multiple replies.

A number of built-in services are available in GEAR and it can be coupled together to form a new server which provides a new service with possibly additional features. The hierarchy of the servers and the services that they provide is shown in figure 4.8.

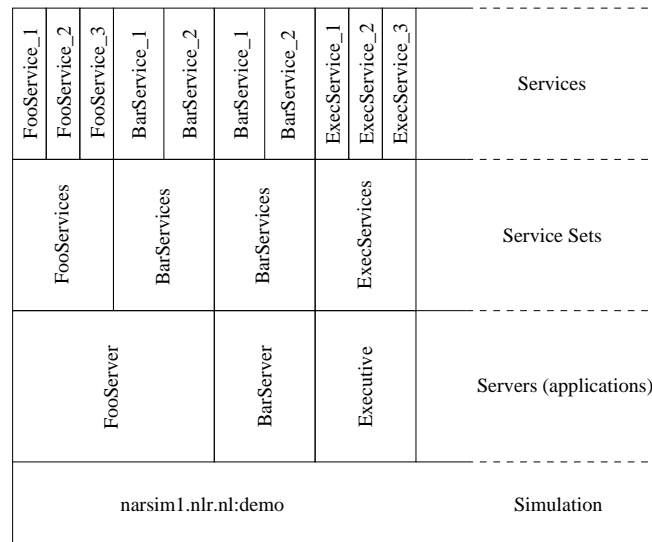


Figure 4.8. Hierarchy in GEAR Middleware

Using a subscription mechanism, there is no need for each server to know what services are provided by which server. It is the job of the *executive server* to maintain and provide information on services available in the system. This information consists of the name and network address of servers including the services that can be provided. To control the progress of a simulation, a *supervisor server* has to be started as well. It mainly provides an overview to the experiment leader about the status of a simulation system. Through its GUI interface, an experiment leader can start, stop or pause the simulation.

After a server is started, it is not yet part of the simulation. It has to register itself (the type of services that it provides) to the executive server which maintains and provide a central administration of the servers and services available. To monitor the global simulation system, a display containing the state of individual server is provided to the experiment leader by the supervisor server so that he/she has an overview and control to the running simulation system.

These two servers are required to be started in each TRS simulation. Having these services as a simulation coordinator, a new service can be added to the existing system by creating a new server or using other existing servers and attach it to the system via registration with the executive server.

4.2.2 Configuration of the TRS System

As TRS is based on the GEAR middleware, the configuration of the TRS system can be decomposed into servers which interact via services defined in their API. During the interaction among servers, the GEAR middleware will arrange all the data exchange such that all the requests can be served by the relevant servers and sent back to the subscribed client. The executive server together with the supervisor

server will manage and control the monitoring, simulation date/time, timer, speed and state control of the TRS simulation system.

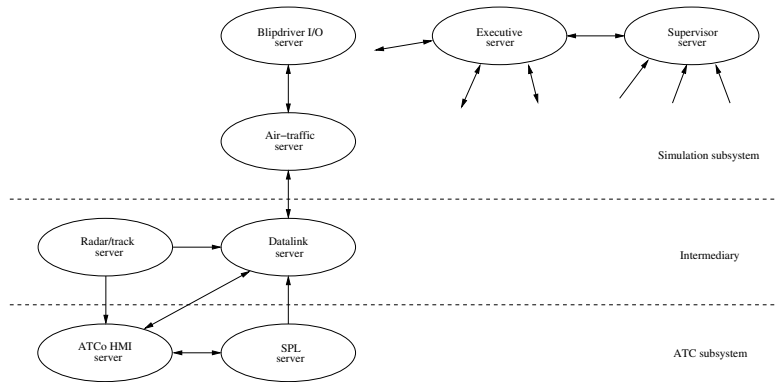


Figure 4.9. A Possible TRS System Configuration

Figure 4.9 shows one example of the architecture of the TRS simulation system, constructed from several servers each providing some services. The servers shown have the following functionalities:

- **Executive server**, has several functions such as management of the simulation, providing of server and service information to the interested servers and control of time during the simulation.
- **Supervisor server**, provides a graphical user interface (GUI) for supervision (by test-supervisor / experiment leader) of the simulation as shown in figure 4.10. It provides commands to start and stop the simulator and also facilities to control the simulation time and speed (time rate). It also supply facilities for on-line monitoring for keeping the experiment leader informed.
- **Blipdriver I/O server**, which is also called pseudo-pilot Human-Machine Interface (HMI) server, has the function to form a user-interface between the air-traffic server and pseudo-pilot. It enables the pseudo-pilot to give command to the aircraft that is controlled. This server also gives feedback by showing the status of the aircraft under control.
- **Air-traffic server**, contains several flight control, aircraft performance and navigational models such as vertical navigation, interceptions of radial, standard routes, etc.
- **Radar/track server**, converts the position of simulated aircraft into an observed track.
- **Datalink server**, routes the message to the correct pseudo-pilot or Air-Traffic Controller (ATCo) and to simulate realistic delays depending on the simulated network.
- **ATCo HMI server**, a display server which supports several input devices (rollballs, mouse, buttons, TIDs) for the ATCo to input commands.

- **SPL server**, maintains and distributes system plan information.

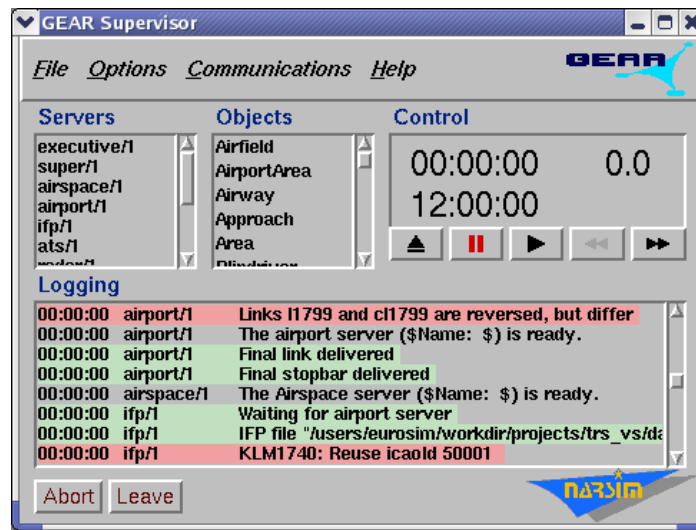


Figure 4.10. GEAR Supervisor

The configuration of the TRS simulation system is highly depending on the requirements imposed on it by the user. To build up a TRS system, the user can choose servers that provide services that are needed or extend the functionality of existing servers. For creating a new service, GEAR provides an Interface Definition Language (IDL) compiler that produces an interface to the other server. Note that the GEAR IDL is not equal to the CORBA IDL.

For the experiment which was developed (described in the next chapter), one particular server which displays the position of the object in the airport map, called *ToHMI server* (Tower HMI) will be used. Figure 4.11 is the display that it gives to the experiment leader.

4.2.3 Simulation states

In a GEAR simulation, there is no global simulation state. The main reason for that is to keep the system as simple as possible. Servers might start at an arbitrary simulation moment. However there is a local state on each server. It is not used by the executive server but can be useful for the experiment leader to get an overview of the system. Those local states are:

- **Start**. After registering itself to the executive server, a server is come into this state. Even if the server does exist, it is not yet actively participated in the simulation
- **Ready**. Whenever an initialization is successful, a server will log a READY message and it enters this state. A experiment leader should not start a simulation before all participating servers enter this state. To check the local states,

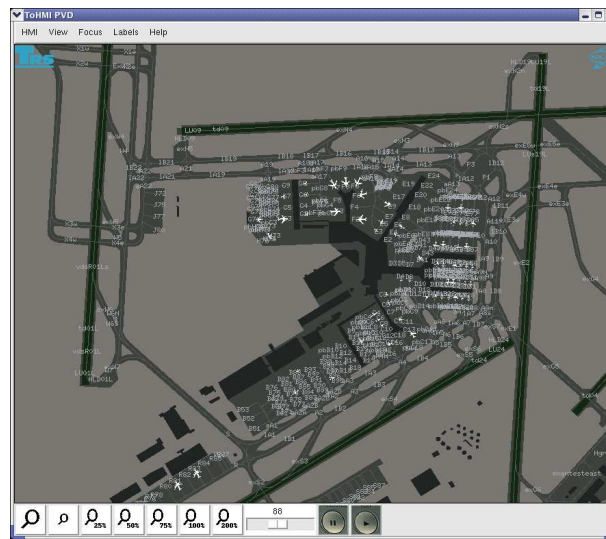


Figure 4.11. ToHMI: Airport Map Display

an experiment leader can observe a color indication in the gear supervisor window as seen on figure 4.10.

- **Exit.** This state is achieved when a server is disconnected from executive server.

4.2.4 Time Management

The executive server maintains and distributes three central clocks, namely the world-time clock (wallclock time), the simulation time and the experiment time, which as the following function:

- *World-time clock* in GEAR is synchronized with the Universal Time (UTC) or also known as Greenwich Mean Time (GMT). It is necessary for getting a reference when a simulation is paused or for maintaining real-time behavior of the GUI.
- *Simulation time*, represents the simulation time which starts from zero and running with a certain simulation speed. By default, the speed is defined to have a value "1".
- *Experiment time*, represents the simulated world-time. It is synchronous with the simulation time but with an arbitrary offset.

4.3 RT CORBA Implementations: The ACE ORB (TAO)

Among several RT CORBA implementations currently available, such as ORBexpress (developed by Objective Interface Systems, OIS), HARDPack (High Availability Real-time Distributed Package-for CORBA, developed by Lockheed Martin Federal System), Visibroker RT (developed by Borland), TAO is the most viable candidates because its performance is relatively higher than others. It is the only open-source RT CORBA implementations which in several studies ([37], [32], [5]) has been compared to the other commercial products mentioned before. Moreover, TAO has also been used in several military CORBA projects of United States Army [32], used as part of the HLA RTI-NG (High Level Architecture - Next Generation RunTime Infrastructure) implementation for DMSO HLA [28], as well as some commercial projects, including the project of Boeing/McDonnell Douglas in real-time avionics mission computing [18]. Many other companies have also used TAO for real-time application in the area of communication systems, particularly telecom, medical, aerospace, and financial services [7].

Even if it is an open-source implementation, TAO is compliant to most of the features and services defined in CORBA 3.0 specification and has been ported in multiple platform (many versions of UNIX (e.g., Solaris 1.x and 2.x on SPARC and Intel, SGI IRIX 6.x, HP-UX 10.x and 11.x, Tru64UNIX 4.x, AIX 4.x and 5.x, SCO) including linux (e.g. Debian Linux 2.x, RedHat Linux 5.2, 6.x, 7.x, 8.x, and 9.x, Timesys Linux, FreeBSD, and NetBSD), several real-time OS (e.g. LynxOS, VxWorks, QnX Neutrino, OS9, and ChorusOS) and several "general-purpose" OS (e.g. WinNT 3.5.x, 4.x, 2000, Embedded NT, XP, Win95/98, and WinCE using MSVC++). Compare to standard CORBA implementation, TAO improves the predictability of its Object Request Broker (ORB) and provides a real-time guarantee by integrating the network interfaces, OS I/O subsystem, ORB, and middleware services for ensuring predictable end-to-end QoS.

In the benchmark investigation by Boeing/McDonnell Douglas [37], TAO was selected to represent an open-source implementation of RT-CORBA and was compared to other commercial RT-CORBA implementations, namely HARDPack and ORBexpress. The assessments include the ORB portability, time required to transfer data of various sizes and types between a single client and a single server, interoperability with other ORB implementation (possibly non real-time ORB) and customer's survey on CORBA features. In most cases TAO and ORBexpress outperformed the HARDPack. The results shows that TAO is comparable to ORBexpress in terms of portability to different platforms and both are having a good performance in delivering basic services.

For comparison of the raw-performance which measures various data transfers between single client and server, ORBexpress is better. However this result measures the general operation of the ORB and not the aspects such as bounded operation times, multi-threading, priority inversion and resource control.

The performance of a real-time system is highly determined by the configuration of the system, which in most cases is system-specific (i.e. no general scenario can be used for comparison). The result from [37] shows that TAO offers more services and more options for ORB configuration to meet the demand of various system-specific requirements. The highly configurability of TAO is an important aspect for the implementation of real-time ORB.

TAO is based on the Adaptive Communication Environment (ACE) framework that implements object-oriented framework for many core patterns for concurrent communication software. Using the patterns defined in ACE, TAO is designed to automate the delivery of high-performance and real-time QoS for distributed applications.

Standard CORBA services are implemented in TAO as well as the additional service targeted for various types of Distributed Real-time Environment (DRE) application domains. Those real-time services are:

- Load balancing service: providing round robin and minimum dispersion algorithms to help balance out the load across a group of machines
- Real-time event service: implementing source and type-based filtering, event correlations, real-time dispatching, and UDP/IP multicast communication. It is the extension of standard CORBA event service
- Scheduling service: for both static (rate-monotonic algorithm) and dynamic (maximum urgency first, [36]) scheduling

Of these services, only two are appropriate in this study, namely the TAO real-time event service and TAO real-time scheduling service. Real-time Event Service will be the main component which implements the RT-CORBA specification while the scheduling service can be embedded as a complementary part of the system. In addition, another TAO service, namely TAO naming service, which is part of the standard CORBA specification, will be used as well. It decouples the connection between peer so that each peer which connects to the system, either as a client or server not necessarily knows each other explicitly.

4.3.1 TAO Real-Time Event Service Configuration

The core element in the event service is defined as an event channel. It mediates the data exchange and provides the interface (proxy) for clients and servers which are connected to the system. In TAO, the exchange of data in terms of events between client and server via the event channel are done in the peer to peer fashion. The one who provides the data acts as a *supplier* and the other one who consumes the data has the role as a *consumer*. Event channel provides proxy interface for the connection of consumer and supplier to it. *Proxy supplier* and *proxy consumer* is

used by the consumer and supplier respectively to connect and disconnect from the event channel.

Each of the supplier/consumer can behave actively or passively in exchanging the data. A passive *push* consumer will just wait for an event pushed to it by an active push supplier. Likewise, an active *pull* consumer will actively pull an event from a passive pull supplier (see figure 4.12). The role as a consumer or supplier is interchangeable, which means that it not a fixed role. A supplier for one service can be a consumer of other service as well, vice versa.

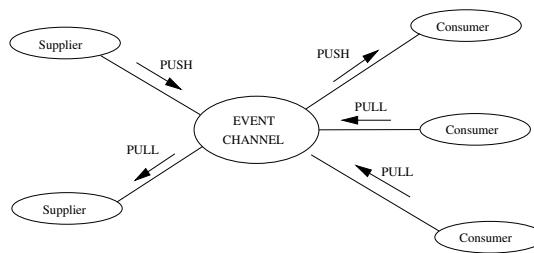


Figure 4.12. Participants in the Event Service

Either consumer or supplier which has a role using pull model will continuously poll the Event Channel. It means that there will be an excessive overhead when there is no event (it will just poll for nothing). In the push model, the event channel will only be pushed by the supplier if there is event(s) available for consumer. There is no overhead in this type of model. Therefore the push model is preferable where the suppliers push the data to the consumers via event channel.

In TAO event service there are several enhancements compare to the event service of standard CORBA. Some of those enhancements are:

- Event filtering: consumers can subscribe to a certain event based on the event type or supplier ID. This feature improves the efficiency event delivery.
- Event correlation: event channel can give notification to consumers while *all events* defined in the event-set have arrived or while *one of event* in the event-set has arrived.

The event channel itself is constructed from several modules as depicted in figure 4.13. It shows how the event channel manages the data flows from supplier to consumer [11]. As figure 4.13 shows, in general there are two main strategies to improve the data transfer efficiency applied by the event channel. The first strategy is to filter events received from the supplier by using *subscription-filtering & event correlation* module so that each event can be transferred to the appropriate consumer. The second strategy deals with the efficiency regarding the priority of which event should be dispatched first to the consumer. This part is divided further into two modules, namely the *run-time scheduler* and *dispatcher*. Run-time scheduler will define which scheduling strategy will be used to arrange the order of the priority

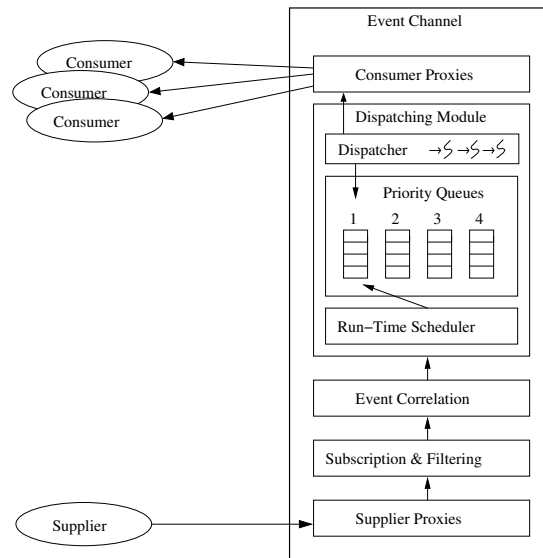


Figure 4.13. TAO Real-Time Event Channel

and the dispatcher will implement an efficient dispatching strategy, based on the priority that has been assigned to each event by the run-time scheduler. These two modules are only necessary if the application requires the real-time performance of the execution. In case where the real-time guarantee is not needed, these two modules can be omitted.

In addition, the TAO event service can also omit the subscription-filtering and event correlation module to provide standard service defined by the standard CORBA specification. The re-configurable property of the event service in TAO provides a flexible option for the application to use the service that it mainly needs and it helps to increase the performance of the ORB.

4.3.2 TAO Real-Time Scheduling Service Configuration

The scheduling service framework in TAO provides several scheduling strategies including *Rate Monotonic Scheduling* (RMS), *Earliest Deadline First* (EDF), *Minimum Laxity First* (MLF) and *Maximum Urgency First* (MUF).

To pass the real-time requirement of the application to the ORB, both the supplier and consumer (conceptually defined as `RT_OPERATION`¹) specify the QoS in the form of `RT_INFO`. It is a data structure consisting the following values to describe the QoS requirement of the application:

- *Criticality*: level of significance of operation's completion prior to its deadline
- *Worst-Case Time*: worst-case execution time

¹all schedulable entity is called `RT_OPERATION`

- *Typical Time*: typical execution time
- *Cached Time*: server data caching time
- *Period*: rate of the event
- *Importance*: an indication of a operation's significance and used as a "tie-breaker" to determine priority when other scheduling parameters are equal
- *Threads*: number of internal threads contained by the operation

RT_INFO structure and the scheduling strategy defined by the user are sent to the scheduler (i.e. run-time scheduler) to be processed. The output of this process are the *Dispatching priority* and the *Dispatching sub-priority* assigned to each event which will be sent to the ORB so that the ORB can put each of those event accordingly in a dispatch. The event which has the highest priority in a queue is dispatched into the proper consumer.

In general, the priority assignment based on the RT_INFO is divided into two steps, *offline* and *online* schedule configuration process. Figure 4.14 shows each steps in the schedule configuration which are described as follows [10]:

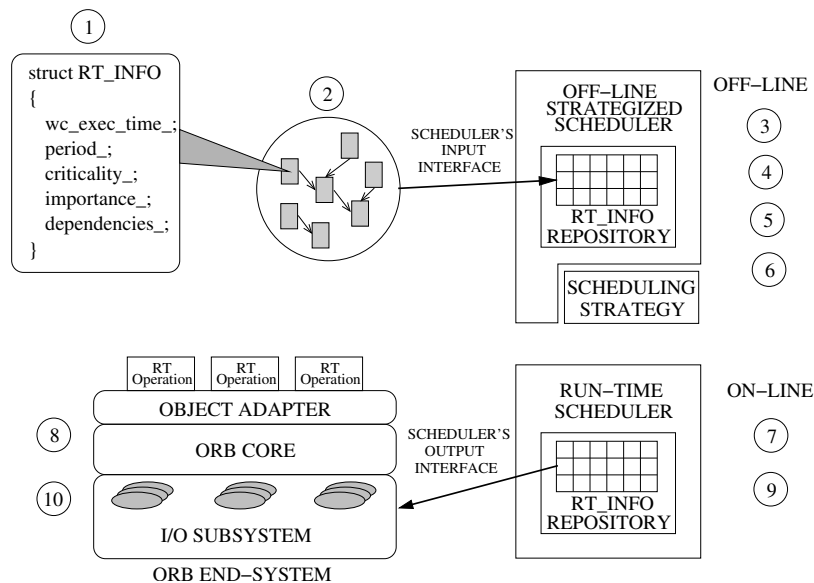


Figure 4.14. TAO Real-Time Scheduling Service [10]

1. User defines the QoS of the application in terms of RT_INFO, including the dependencies between the operations used in the application.
2. Send the QoS information to the scheduling service which will put those QoS inside its repository so that it can construct the operation dependency graphs.

OFFLINE

3. Assesses the schedulability of all the *RT_OPERATION*
4. Assigns static priorities and sub-priorities to *RT_OPERATION*. This assignment is based on the scheduling strategy that is used. For example in static strategy like RMS, it only uses static priority based on the operation criticality. Another dynamic strategy like Maximum Urgency First (MUF) uses sub-priority based on the operation importance and dependencies.
5. Based on the scheduling strategy that is selected by the user, in this step the mapping of static priority and static sub-priority into dispatching priority and dispatching sub-priority are done.
6. Assign dispatching queue configuration.

ONLINE

7. Supply the dispatching queue configuration to the ORB.
8. Configure the queue based on the dispatching queue configuration.
9. Supply static portion of dispatching priority and dispatching sub priority to the ORB.
10. Dynamic queue assign dynamic portion of dispatching subpriority (and possibly dispatching priority).

To further optimize the performance of the ORB, TAO provides different options of ORB configuration that can be used to get an optimal performance of the system. This part is explained more detail in appendix C.

5 CASE STUDY: DISTRIBUTED REAL-TIME SIMULATION OF AIRPORT VEHICLE MOTION

5.1 Introduction

Re-usability and inter-operability are key challenges in present real-time simulator development. This case study will demonstrate the inter-operability of several real-time simulation environments by generalisation of their interfaces. Furthermore an assessment the possibilities for preservation of the real-time execution guarantee will be presented.

As introduced in section 4.2, NLR's TRS (Tower Research Simulator) is mainly designed for airport traffic simulation system. The TRS has a main function is to simulate the real-time air-traffic control activities at an airfield using a visual environment that has a 360° field-of-view projection area where visual cues are simulated highly realistically. Inherently, it is not adequate for real-time simulation of dynamic vehicle motion. For that purpose, EuroSim is more appropriate and it has been applied coupled to TRS in this case study. This interconnection will demonstrate the *inter-operability* among different real-time simulation systems.

The middleware functionality included in TRS is not equipped with the mechanism to guarantee real-time performance of the simulation system. To incorporate a real-time guarantee mechanism into the simulation system RT-CORBA is used. Moreover, by using a standardized interface like RT-CORBA, the interface of the simulation system can be generalized to support *re-usability* for legacy simulation application.

As an initial demonstrator, a simple moving vehicle model is built and executed using EuroSim. This simple simulator produces the coordinate and the orientation of the vehicle where the input is given interactively by the user. This output data from EuroSim will be communicated to the TRS simulation environment.

In this scenario, the vehicle model is considered as a fire-truck which is controlled via EuroSim and driven over the airport that is simulated with TRS. It is assumed that the vehicle is moving in a flat-ground airfield so that only two dimensional (x-y) coordinates and orientation (yaw-angle²) are defined. This movement is controlled by the Simulation Controller (one of the elements of the EuroSim simulator as explained in section 4.1.5) which provides the GUI-button to the user to steer the vehicle and to change the simulation states (e.g. start, pause and stop the

²Yaw-angle is defined as the motion or rotation of the vehicle around its normal axis. It determines the left-right movement of the vehicle

simulation).

5.2 Vehicle Model

The vehicle model used in this experiment is based on the model from [39]. This model is composed from several elements as shown in figure 5.1.

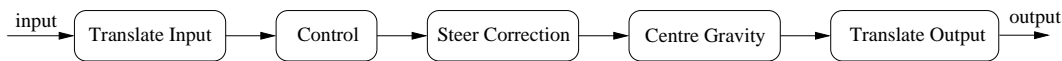


Figure 5.1. Vehicle Model

1. Translate Input, defines the input signal to the model as a result from the input given by the user.
 - Input: gearbox, throttle, brake and steer signal. All of these input signals can be given by the user using the GUI button interface from EuroSim simulation controller as explained in the next section.
 - Output: steer, brake, clutch, acceleration (all of these output are structured in a variable called control signal). The "brake" and "acceleration" are having the value either "1" or "0" to indicate the "press" and "release" of the respective pedal. The "steer" value is given in radian with the range [0.0 1.0].
2. Control, defines the behavior of the model based on the user's input. For example, this component will synchronize the behavior of the acceleration pedal and the brake pedal.
3. Steer Correction, changes the forward and lateral chassis orientation of the vehicle based on the given steer angle.
4. Center Gravity, transfers the control signal to global x and y position values (as a function of time) of the center gravity point of the chassis. The other parameters like velocity, acceleration and orientation are derived from x-y position values.
5. Translate Output, convert the value for displaying purpose.

All of these models are implemented in a C-code and they are compiled and executed in a EuroSim simulation system. All of these model is synchronizd with the 40 Hz (i.e. 25 ms) clock as shown in figure 5.2, which means that data update to each component is done every 25 ms.

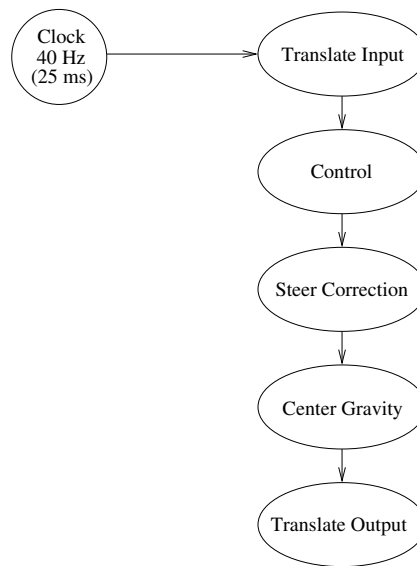


Figure 5.2. Scheduling of EuroSim Simulation Component

5.3 The simulation scenario

This case study investigates a scenario where a standard airport operation is simulated with TRS in which two vehicle objects (considered as fire-trucks of which the motions are controlled via EuroSim) are added. Each fire-truck model is executed and controlled by EuroSim which was installed on two different machine architectures, namely PC linux-x86 (RedHat 8.0) and SGI-IRIX 6.5.

Via the GUI on EuroSim Simulation-Controller, several steering buttons are provided to the user, namely, steering angle with range [0.0 1.0], acceleration of the vehicle [0 1], and brake [0 1] to stop/run the vehicle. This GUI interface is shown in figure 5.3. The value "1" and "0" of the Acceleration means that acceleration pedal is "pressed" and "released" (i.e. constant speed/no acceleration), respectively. The same representation is applied for the "brake" button. The other button ("steering angle"), represents the steering angle that will change the orientation of the vehicle, either to left or right (based on the Yaw-angle). To change the orientation of the vehicle, user can press one of the steering button shown in figure 5.3. It steers the vehicle to move around. For immediately steering the vehicle straight, user has to press the "steer 0.0" button.

Each EuroSim session produces the (x-y) coordinate and the orientation (yaw-angle) of the vehicle. As the vehicle is moving, the coordinate and orientation is updated according to the update frequency of 40Hz which was initially defined on the EuroSim Schedule Editor (figure 5.2). This "periodically-changed" data in turn is communicated to the TRS airport map display which can be extended as well to the TRS 360° projection screen. The airport map display shows the position of each vehicle on the airport (in this case Schipol airport). As the data from EuroSim are periodically updated, the vehicle data shown in the airport map display will keep

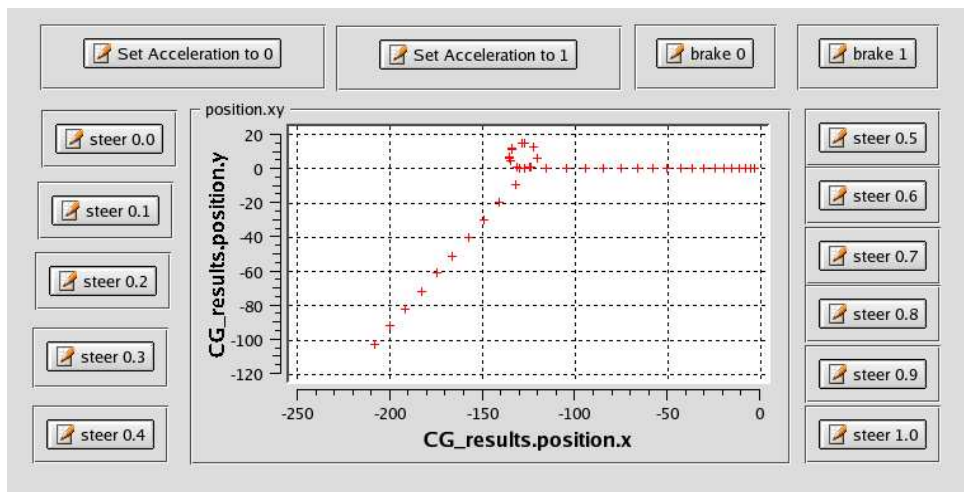


Figure 5.3. Simulation Controller: GUI Button for Vehicle Control & Steering

moving and change coordinate/orientation according to the update frequency. Figure 5.4 show the position of the vehicles, controlled by EuroSim on the TRS airport map displays. In that display, the vehicles are shown as white rectangles moving in the airport.

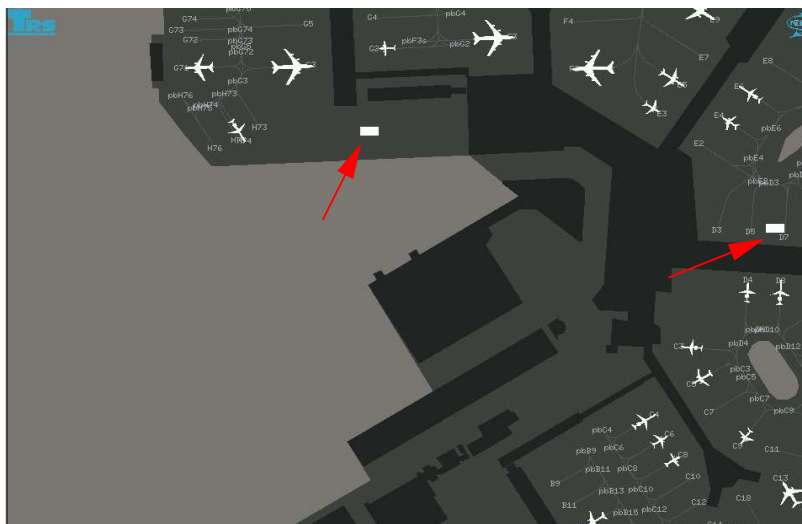


Figure 5.4. Two Vehicles on TRS Airport-Map Display

5.4 Distributed Simulator System Implementation

The global overview of the implementation of the distributed system is described in figure 5.5. Eurosim daemon, the core of EuroSim simulator, is a daemon for initializing the EuroSim simulation process and for configuring the target hardware. A EuroSim process might be the process which executes the `extSimAccess` or it can be the process which executes the Simulation Controller (`extSimAccess` is also

referred to as XSA library). Each process that uses EuroSim's properties will always contact the daemon for acquiring authorization regarding the license key to use the EuroSim product. Once this authorization is given by the daemon in the initialization procedure, the `extSimAccess` can establish the connection to other processes (not necessarily in the same machine) using the RPC-based protocol.

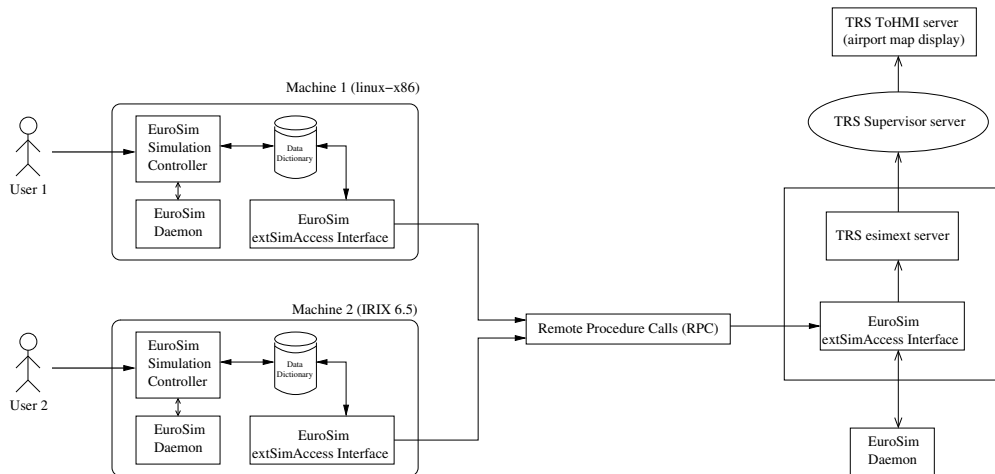


Figure 5.5. Study Case: Implementation of the distributed system

As shown in figure 5.5, this scenario uses 2 EuroSim sessions, running on different machines where each session executes an instantiation of the vehicle model. The information regarding the position and orientation of these two vehicles are sent to the TRS through `extSimAccess` library. On machine 3, once an updated data is arrived, it will trigger the event-handler which in turn informs the TRS server to update its data.

EuroSim provides an ad-hoc solution for interfacing EuroSim simulation with other external simulators. The library called `extSimAccess` from EuroSim (also referred as eXternal Sim Access library / XSA library) provides the communication functions based on the RPC protocol [13]. First the user has to define the data that should be shared including its permission access (read or write) to the data dictionary (internal to EuroSim) and after that he/she should provide those data to the `extSimAccess` library. When there is an update to those shared data, an event will be generated by the EuroSim Simulation Controller so that the event-handler can take appropriate action which is related to the state of simulation.

Using this EuroSim XSA library, a distributed scenario can be implemented. However it is required that each machine knows each other explicitly by having an IP address of the other machine. For the purpose of demonstration, each machine has a different platform. One machine has linux-x86 platform and the other machine is using IRIX platform. As there is different data representation (little-endian and big-endian byte ordering for linux and IRIX respectively) between these 2 architectures, user has to take care of this conversion so that each machine receives the appropriate data.

EuroSim is designed to be a real-time simulator that provides hard real-time guarantee for the execution of its simulation model. However if data from external sessions are used and those data are transferred via the `extSimAccess` library, there will be no hard real-time guarantee anymore. EuroSim can only schedule the execution of the models which are running in one session. EuroSim is not designed for providing hard real-time guarantee for a simulation running on a distributed system. In such a distributed scenario, user can still expect the soft real-time performance which fully depends on the speed of the network and the machine.

5.5 Distributed Real-Time Simulation System Using TAO

In the previous section it was shown that the use of external data in EuroSim deletes real-time guarantee. In order to preserve this real-time guarantee in distributed real-time simulations, a real-time enabled middleware like TAO can be used for the coupling of EuroSim to external simulators. In addition, it is beneficial to generalize the interface for the interconnection among several simulators so that the data conversion is not necessary anymore. For this case study that uses the EuroSim and TRS simulation systems, the ORB of TAO is implemented as their interface. This interface is independent from the underlying platform that is used, namely the operating system and data representation in which the EuroSim and TRS are installed. This can be seen in figure 5.6.

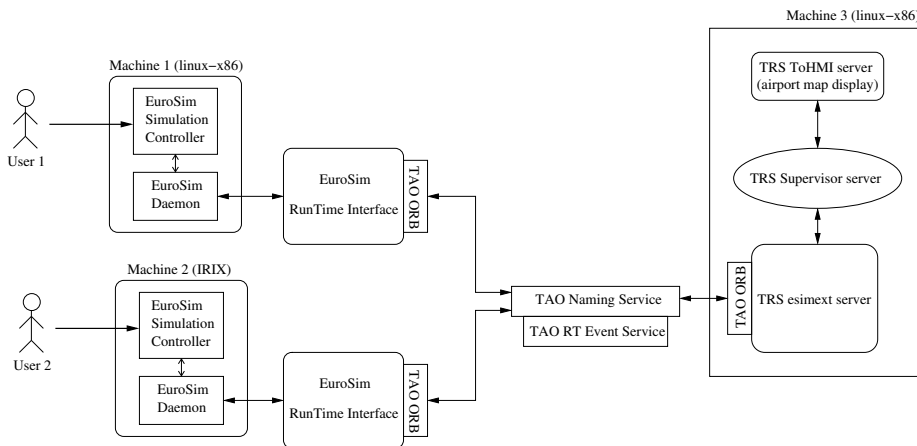


Figure 5.6. EuroSim-TAO-TRS System Configuration

As figure 5.6 shows, TAO Naming Service is used as a mediator to register all the machines that are involved in the simulation system, either as a supplier (EuroSim) or a consumer (TRS). TAO Real-Time Event service which is embedded³ to the TAO Naming Service is used to manage the data exchange such as event subscription

³TAO Real-Time Event Service will also contact TAO Naming Service so that supplier or consumer do not have to know the explicit location where the Event Service is executed

by the consumer, event filtering and event correlation. It does so by providing the event channel to the consumer and supplier as a point of contact for exchanging data.

Initially, the EuroSim sessions on machine 1 and machine 2 act as a suppliers of events (vehicle coordinates and orientation) to the TRS airport-map display, which consumes the event and displays the position of the vehicle. However, it is also possible that the roles are changed. For demonstration purpose, the TRS supervisor server is configured to be able to change the state of the simulation system by sending back a state change to the EuroSim sessions on machine 1 and 2 (for example "start", "stop" signal). It implies that in such situations the EuroSim session is the consumer of the events (simulation state) supplied by the TRS supervisor server.

Figure 5.6 shows that TAO Naming Service is used as a center of data exchange so that there is no need for each machine involved in the system to know each other explicitly. Any application that needs certain data can be attached to the naming service to subscribe for that particular data. This way the configuration in figure 5.6 can be generalized by using other simulators. While the TAO Naming Service provides the ID inventory of each machine that is connected to it, TAO Real-Time Event Service will take care of the subscription mechanism between supplier and consumer of events.

Each event (e.g. vehicle coordinate) produced by the EuroSim sessions on machine 1 and machine 2 in figure 5.6 is tagged using a certain identifier so that the consumer (TRS airport-map display) can distinguish which event has to be passed to which vehicle. In the opposite direction, where the TRS supervisor server sends the state changes to both EuroSim sessions, it specifies that event (i.e. simulation state) as a generic event so that any consumer attached to the event channel will receive it. The event filtering and event correlation feature can help to increase the performance of data exchange because one simulation component will not receive unnecessary events/data.

5.6 Results

In this section, the results of the distributed real-time simulation system of the airport scenario case study are presented. It considers both implemented architectures, namely using EuroSim XSA (External Simulation Access) library (referring to figure 5.5) and using RT-CORBA implementation / TAO (referring to figure 5.6). First the data transferred by EuroSim session will be compared to the data received by the TRS to assess the correctness of the data transfer. The next result will assess the speed of the data transfer from EuroSim session to TRS. Figure 5.7 shows the general configuration of the distributed real-time simulation developed in this experiment.

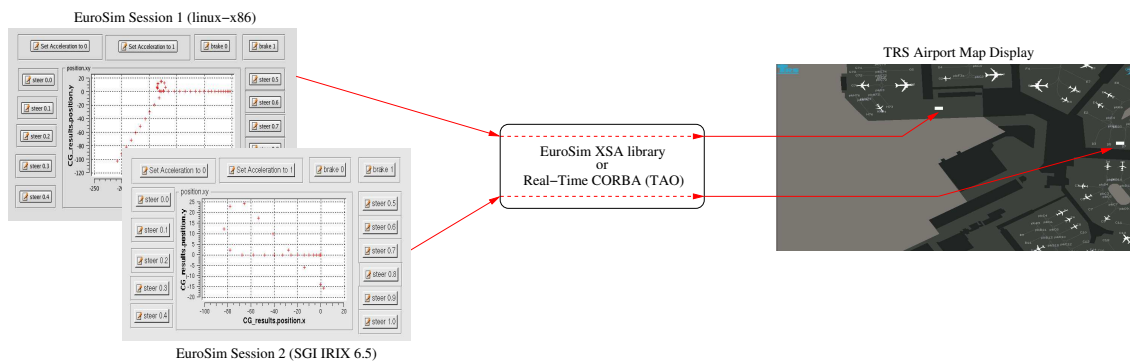


Figure 5.7. General Configuration of Distributed Real-Time Simulation using EuroSim, TRS, XSA library and TAO

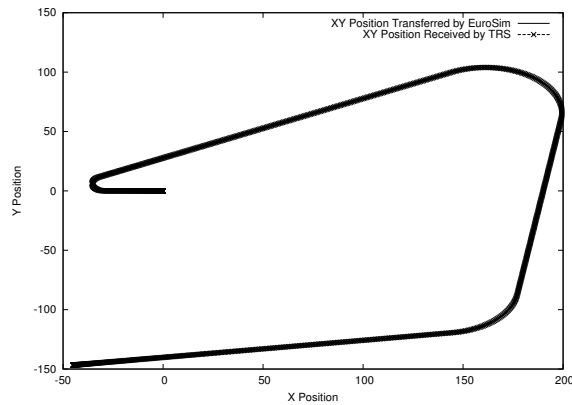
5.6.1 Data Transfer Verification

The basic required functionality of distributed real-time simulations is the ability to correctly exchange data between simulation environments. There must be a guarantee that the data transferred by one side will be received exactly at the same value by the other side in real-time. For assessing the correctness of this data transfer, the recorder in EuroSim Simulation Controller will be used to record the coordinate data produced by EuroSim session during the movement of the vehicle. This data in turn will be compared to the data received at the Airport Map display of TRS. The results are shown in figure 5.8.

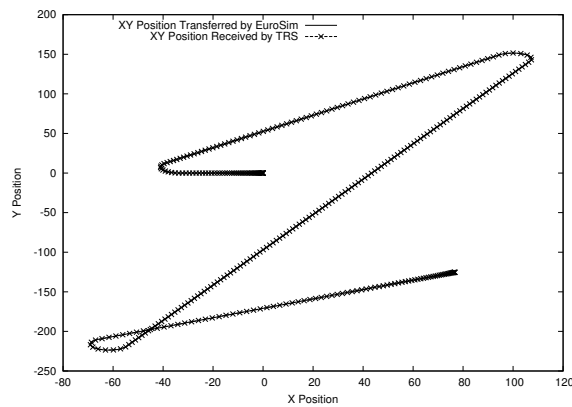
Figure 5.8(a) shows an example of the plot of xy coordinates during one pre-scribed motion of the vehicle. The vehicle initial position is at the point (0,0). Once the acceleration is set to "1", it starts to move. Approximately at the point (-40,0), the steering is set to 0.6 value and immediately set back to 0.0 again so that the vehicle can moves straight again. The next two turns in figure 5.8(a) shows the same sequence but using different steering angle.

As shown in that figure, the data transferred by EuroSim simulation controller is recorded as well as the data received at the TRS server. The overlapping lines in this plot (which uses EuroSim XSA library for inter-connection between Eurosim sessions and TRS server) shows that the data transferred by EuroSim are exactly the same as the data received by TRS.

To demonstrate the capability of steering the vehicle in various kinds of movement, different trajectory is used for the case the TAO is used for the inter-connection between EuroSim sessions and TRS server. As in the previous case, the vehicle is initially has (0,0) position and different steering values is applied to make different vehicle's turn-around movement (as shown in figure 5.8(b)). As in the previous case, this set-up also shows an overlapping trajectory of the data transferred by EuroSim simulation controller and data received by TRS server. For both measurements, the x-y coordinates are recorded up to 4 decimal places.



(a) Inter-connection using XSA



(b) Inter-connection Using TAO

Figure 5.8. The XY Coordinate Track of Vehicle Movement

These results shows that both EuroSim XSA library and TAO give realible data transfer in the sense of the correctness between the data transfered and the data received. In the next section, the time needed for transferring those data will be measured.

5.6.2 The Latency of Time Data Transfer

The latency is defined as the average amount of delay observed by a client from the time it sends the request to the time it completely receives the response from a server [30]. In other words, it is a measure of roundtrip time of data transfer between 2 machines.

The distributed real-time simulation system which has been built, consists of at least 3 different machines which have different time clock. To measure the time it takes

for transferring data from one side to the other side, all of these machines have to be synchronized up to a certain accuracy to get a representative timing measurement. To avoid the time synchronization, the roundtrip time will be measured under the assumption that it takes approximately the same amount of time for data to be transferred in both directions.

The measurement is done by sending a data from a machine running on SGI-IRIX 6.5 platform to another machine running RedHat linux 8.0. The timing function used is `gettimeofday` and it is used to stamp the time just before the data transfer and the time just after the data is received back. The difference between these two timing is considered as latency. This measurement is done for 1000 times of data transfer and using several data sizes for each experiment, as shown in table 5.1 and 5.2.

	2 bytes	4 bytes	8 bytes
Average	29961.978	29959.675	29983.180
Minimum	22576.000	23186.000	23462.000
Maximum	37447.000	35351.000	58769.000
Standard Deviation	597.422	572.908	1120.163

Table 5.1. Time Latency (in microseconds)for Inter-connection using XSA for 1000 samples

	2 bytes	4 bytes	8 bytes
Average	9199.047	9031.125	9048.475
Minimum	7874.000	7558.000	7650.000
Maximum	17452.000	32875.000	17291.000
Standard Deviation	929.098	1237.455	868.847

Table 5.2. Time Latency (in microseconds)for Inter-connection using TAO for 1000 samples

From both tables, it can be seen that the latency for inter-connection using TAO is significantly lower compared to the one which uses XSA. However, the configuration of TAO in this experiment does not include the Real-Time Scheduling Service yet due to the lack of documentation and incomplete implementation of this service in TAO. By incorporating such a scheduling service, it can be expected that the the guarantee of timing consistency can be provided which is important especially in case where there are more data transfer from many EuroSession sessions.

Looking at the maximum value from both tables, it can be concluded that during the measurement there are some interferences from other processes running in the same machines. This can not be avoided since the timing function used (`gettimeofday`) is measuring the wallclock time but not taking into account that process which is being measured might be pre-empted by other process. More appropriate measure is by using `times` function which measures the amount of CPU time used per process. However since the data transfer involve several processes which might be executed

on different machines, the use of `times` is quite difficult. Even if the measurement of round-trip time using `gettimeofday` is not that accurate, by taking more samples the result of the measurement will be representative as well.

Referring to the update frequency in figure 5.2 (25 msec) of each simulation component, both tables shows the possibility to transfer data before the next data update (assuming the same one-way data trip, on XSA needs 15 ms while TAO needs approximately only 4.6 ms). For distributed real-time simulation system, the sooner data can be transferred the better performance can be achieved. Especially in case where there are more simulation components exchange data at relatively the same time. As explained in section 4.3.1, TAO provides an event correlation mechanism that can improve the performance of data exchange between simulation components. In case where the scheduling service is also incorporated, the guarantee of real-time performance can be given as well (in case there is a viable schedule).

6 DISCUSSION & CONCLUSIONS

In this report a case study has been presented in which a distributed real-time simulation system was constructed, integrating several real-time simulators using standardized interface and preserving the real-time guarantee of the system performance. This was achieved by using TAO as a middleware which provides either additional real-time services (for example Real-Time Event Service, Real-Time Scheduling Service), or new modules (for example RT-ORB, RT-POA) which both improve the predictability and control over the management of the resources.

It was demonstrated that the common interface, provided by TAO ORB, generalizes the way in which the simulation environments are inter-connected. The EuroSim simulation environment can be seamlessly integrated with the TRS simulation environment, which not only demonstrates the inter-operability feature but also the preservation of the real-time guarantee. From the simulation result, the inter-connection using TAO is a lot faster than using EuroSim XSA library (RPC-based). It is an important advantage for distributed real-time simulation systems, regarding its stringent time-deadline. However, current state of technology is not yet sufficiently mature to easily allow that integration.

TAO provides support for many different platform which include various version of UNIX and linux. For linux platform, the installation can be done relatively easy, but for UNIX platform, for example SGI IRIX platform, the porting of TAO for this platform is still an ongoing process. However, most of the main feature can already be used & operational.

TAO also provides many different services which are based on the design-pattern framework which allow for flexible configuration and customization. This is an important feature for a middleware used in real-time application as for most cases an optimal configurations of real-time application are system-specific [6]. The current status of TAO is also still under heavy development and improvements are made quite rapidly. For using TAO as an open-source requires considerable amount of time as for some services (for example scheduling service), there are no detail manual available. The main information that can be obtained is the source-code itself. For overcoming the shortage in manual, TAO's developer provides a commercial service for that.

In distributed real-time simulation, a generic interface is needed so that any legacy simulation environment can be possibly integrated into it. This will support the key challenge of current distributed real-time simulation system, namely the re-usability and inter-operability. TAO can be used to provide such a generic interface

6.1 Discussion

The inter-ORB communication in distributed real-time simulation systems needs support from the transport layer protocol [38] to provide a reliable packet delivery mechanism. By default, TAO uses Internet Inter-ORB Protocol (IIOP) which is a TCP/IP-based protocol. As a TCP/IP protocol is in itself a connection-oriented type of protocol which provides mechanisms to guarantee the delivery of every data packet (such as error-control, flow-control, sequencing mechanism), it can introduce large delay and jitter which severely limit end-to-end throughput. The cost of real-time performance in this case is just too high for the gain in reliability [20]. Therefore, to further increase the real-time performance of the inter-ORB communication of the distributed real-time simulation system (which depends also on the transport protocol that is used), a real-time transport protocol is important to be considered. For that purpose, TAO also supports a feature called pluggable protocol, which can include in general any real-time protocol that is appropriate for the application [27]. The incorporation of such a protocol was out of scope for this study.

As RT-CORBA also makes use of threads to execute tasks (application threads), in some situations there is no way to prevent hardware interrupt and Operating System (OS) interrupt handlers from pre-empting application threads. If this happens, there will be another source of overhead introduced to this distributed real-time simulation system [30]. Moreover the predictability of the system is decreasing since there is no knowledge about when the hardware interrupt will finish its task. In such situations the use of a Real-Time Operating System is sensible in order to enhance the predictability of the system. A Real-Time Operating System can assure a consistency of predictability while executing distributed real-time applications [30].

6.2 Conclusions

1. A distributed real-time simulation system has been developed, integrating two real-time simulation systems (namely Eurosim and the Tower Research Simulator/TRS) with a RT-CORBA based middleware system (The ACE ORB: TAO). This system has been used in a real-time simulation of a airport vehicle motion scenario.
2. It has been demonstrated that the integration of several real-time simulation systems as components of the above mentioned distributed real-time simulation system is platform independent. Two EuroSim sessions have been executed on two different operating system (namely linux RedHat 8.0 and SGI IRIX 6.5) while both are sending data in real-time to the TRS simulation system via TAO.
3. The performance of TAO is significantly higher than EuroSim XSA library which is an RPC-based protocol.

4. To preserve real-time guarantee on the distributed real-time system in general, it is necessary that some mechanism exists to provide a way for the user to impose the timing requirement of his/her application. Real-Time CORBA provides such mechanism for its scheduling service in terms of Quality of Service that can be determined by the user. This mechanism does not yet exist in current High Level Architecture specification.
5. To further improve the real-time performance, it is sensible to consider real-time operating system and real-time network protocol.

6.3 Recommendations

For the case study presented in this report, an appropriate scheduling strategy still needs to be assessed since an optimal performance of the distributed real-time simulation system is also dependent on the scheduling strategy that is used. Several scheduling strategies can already be used in case the scheduling service is incorporated in the configuration of TAO ORB [10].

Using TRS as part of the simulation system environment, many scenarios can be derived to simulate the needs of scheduling service that can help to provide guarantee of real-time execution (such as incorporating more fire-trucks and schedule them efficiently to extinguish the flame in a fire accident at an airport). As has been done in the Simultaan project [8], the simulation of the fire-truck can be relatively easy to be extended to the 3D visualization of the fire-truck model moving around in the Schiphol airport.

Bibliography

- [1] IEEE Std 1516. IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA), September 2000.
- [2] Steffen Straßburger. *Distributed simulation based on the high level architecture in civilian application domains*. Ph.D. dissertation, Otto-von-Guericke-Universität Magdeburg, April 2001.
- [3] Judith S. Dahmann. The high level architecture and beyond: technology challenges. In *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 64–70. IEEE Computer Society, 1999.
- [4] Dennis Noll David Corman, Jeanna Gossett. Experiences in a distributed, real-time avionics domain-weapons system open architecture. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. IEEE Computer Society, 2002.
- [5] United States Department of Defense. Defense Infrastructure Information/Common Operating Environment (DIICOE). DII COE design review for real-time CORBA products. Available on the Internet: http://diicoe.disa.mil/coe/aog_twg/twg/rttwg/Baseline/DesignReview/RTCORBA/, 2000.
- [6] Christopher D.Gill and Douglas C.Schmidt. Multi-paradigm scheduling for distributed real-time embedded computing. *Proceedings of IEEE Special Issue on Modeling and Design of Embedded Systems*, 91(1), January 2003.
- [7] University of California Irvine Distributed Object Computing Group. Tao overview. Available on the Internet: <http://www.cs.wustl.edu/schmidt/TAO-overview.html>, 2004.
- [8] Nationaal Lucht en Ruimtevaartlaboratorium, TNO, Delft Aerospace, Siemens, Fokker Space BV, Hydraudyne Systems Engineering BV, and Fokker Space BV. Simultaan distributed training simulators. Available on the Internet: <http://www.nlr.nl/public/hosted-sites/simultaan/index.htm>, 2000.
- [9] Richard M. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley & Sons, Inc., 2000.
- [10] Christopher D. Gill, David L. Levine, and Douglas C. Schmidt. The design and performance of a real-time CORBA scheduling service. *Real-Time System*, 20(2):117–154, 2001.
- [11] Timothy H. Harrison, David L. Levine, and Douglas C. Schmidt. The design and performance of a real-time CORBA event service. In *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 184–200. ACM Press, 1997.

- [12] Object Computing Inc. *TAO Developer's Guide Volume 1 & 2*. Object Computing Inc., 2003.
- [13] Sun Microsystems Inc. RPC: Remote Procedure Call Protocol specification. Available on the Internet: <http://rfc.net/rfc1050.html>, April 1998.
- [14] David A. Karr, Craig Rodrigues, Joseph P. Loyall, Richard E. Schantz, Yamuna Krishnamurthy, Irfan Pyarali, and Douglas C. Schmidt. Application of the quo quality-of-service framework to a distributed video application. In *Proceedings of the Third International Symposium on Distributed Objects and Applications*, page 299. IEEE Computer Society, 2001.
- [15] E. Kessler. Deploying networked real-time simulation, putting the virtual enterprise to work some aerospace experiences. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, page 285b. IEEE Computer Society, 2002.
- [16] Yamuna Krishnamurthy, Vishal Kachroo, David A. Karr, Craig Rodrigues, and Douglas C. Schmidt. Integration of qos-enabled distributed object computing middleware for developing next-generation distributed application. In *Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, pages 230–237. ACM Press, 2001.
- [17] United States Naval Air Warfare Center-China Lake. Open systems architecture puts six bombs on target. Available on the Internet: <http://www.cs.wustl.edu/schmidt/TAO-boeing.html>, May 2004.
- [18] David L. Levine, Christopher D. Gill, and D. Schmidt. Dynamic scheduling for avionics applications. In *Proceedings of the 17th IEEE/AIAA Digital Avionics System Conference*, 1998.
- [19] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.
- [20] Jane W.S. Liu. *Real-Time Systems*. Prentice Hall PTR, 2000.
- [21] Raymond Michiels. A survey of the NARSIM C/S Middleware. Technical Report NLR Technical Publication no.637.406, Nationaal Lucht- en Ruimtevaart Laboratorium, January 2000.
- [22] Jeff Nielsen. Challenges in developing the JTLS-GCCS-NC3A Federation. *the Fall Simulation Interoperability Workshop*, September 1998.
- [23] United States Department of Defense (DoD). High Level Architecture Interface Specification, version 1.3, April 1998.
- [24] Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification, revision 2.3. Available on the Internet: <ftp://ftp.omg.org/pub/docs/formal/98-12-01.pdf>, March 1998.

- [25] Object Management Group (OMG). RealTime-CORBA specification, version 2.0. Available on the Internet: <http://www.omg.org/docs/formal/03-11-01.pdf>, November 2003.
- [26] Object Management Group (OMG). Catalog of specialized CORBA specification. Available on the Internet: http://www.omg.org/technology/documents/specialized_corba.htm, March 2004.
- [27] Carlos O’Ryan, Fred Kuhns, Douglas C. Schmidt, Ossama Othman, and Jeff Parsons. The design and performance of a pluggable protocols framework for real-time distributed object computing middleware. In *IFIP/ACM International Conference on Distributed systems platforms*, pages 372–395. Springer-Verlag New York, Inc., 2000.
- [28] Carlos O’Ryan, Douglas C. Schmidt, and J. Russell Noseworthy. Patterns and performance of a CORBA event service for large-scale distributed interactive simulations. *International Journal of Computer Systems Science and Engineering*, 17(2), March 2002.
- [29] Tod Reinhart, Carolyn Boettcher, G. Andrew Gandara, and Mark Hama. Defining a security architecture for real-time embedded systems. *Command and Control Research and Technology Symposium (CCRTS)*, June 2004.
- [30] D. Schmidt, M. Deshpande, and Carlos O’Ryan. Operating system performance in support of real-time middleware. In *Proceedings of the IEEE Workshop on Object-oriented Real-time Dependable Systems*. IEEE Computer Society, 2002.
- [31] D. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale. Alleviating priority inversion and non-determinism in real-time CORBA ORB Core architectures. In *Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*, page 92. IEEE Computer Society, 1998.
- [32] Edgar V. Shrum. Use of RT CORBA in the U.S. Army. In *Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. IEEE Computer Society, 2001.
- [33] Dutch Space, Nationaal Lucht en Ruimtevaart Laboratorium, and Atos Origin. EuroSim Mk3.0 Software User’s Manual. Available on the Internet: <http://www.eurosim.nl/news/SUM-Mk3.0.pdf>, 2003.
- [34] Dutch Space, Nationaal Lucht en Ruimtevaart Laboratorium, and Atos Origin. EuroSim real-time simulator. Technical report, Dutch Space and Nationaal Lucht en Ruimtevaart Laboratorium and Atos Origin, 2003. Available on the Internet: <http://www.eurosim.nl>.
- [35] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.

- [36] D. B. Stewart and Pradeep Khosla. Real-time scheduling of sensor-based control systems. In *IEEE Workshop on Real-Time Operating Systems and Software (RTOS '91)*, pages 144–150, May 1991.
- [37] Phantom Works Advanced Information Systems. The real-time CORBA trade study. Available on the Internet: <http://www.ois.com/resources/corb-10-download.asp>, January 2000.
- [38] Andrew S. Tanenbaum. *Computer Networks; 4th edition*. Prentice Hall, 2003.
- [39] A.A. ten Dam, R.P.J.Groothuizen, W.J.Vankan, R.van Sterkenburg, and W.F.Lammen. Conservering van de status van de capaciteiten van de behaviour model component van het simultaan project. Technical report, Nationaal Lucht-en Ruimtevaart Laboratorium, January 2000.
- [40] Brad Fitzgibbons Thom McLean, Richard Fujimoto. Next generation real-time RTI software. In *Proceedings of the 5th IEEE International Workshop on Distributed Simulation and Real-Time Applications*. IEEE Computer Society Press, 2001.
- [41] T.Usländer. Opera: A CORBA-based architecture enabling distributed real-time simulations. Available on the Internet: <http://opera.iitb.fhg.de/servlet/is/114/isorc99.pdf?command=downloadContent&filename=isorc99.pdf>, 1998.
- [42] Steve Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), february 1997.
- [43] R.H.de Vries, J.Keijzer, F.van Lieshout, A.A.ten Dam, and J.M.Moelands. Enhancements to EuroSim. In *proceedings of the 7th International WorkShop on Simulation for European Space Programmers (SESP)*. European Space Agency (ESA), November 2002.
- [44] W.Lammen, A.A.ten Dam, W.W.M.Heesbeen, and R.Franco. Mosaic: Automated model transfer between cots tools and standards. In *proceedings of the 6th International WorkShop on Simulation for European Space Programmers (SESP)*. European Space Agency (ESA), October 2000.
- [45] A. Zabek, L. Feinerman, and D. Seidel. Joint warfighting program information superiority experiment trailblazer. In *Proceedings of th Fall 1998 Simulation Interoperability Workshop*, September 1998.

Appendix A

High Level Architecture Components

A.1 HLA Interface Specification

As mentioned before, there is a separation of tasks between RTI and federate. Any simulation specific data is limited within the scope of each federate only. There are six classes of services that describes the way RTI and federates communicates with each other, which are specified in the HLA Interface Specification.

- Federation Management, defines the coordination of federation-wide activities during a federation execution in terms of existence and membership for each federates. This service provide a way to create/destroy federation execution, join/resign of federate to/from the federation, and maintaining the states of federation execution.
- Declaration Management, defines a way for federates to express its intent for publish or subscribe data to and from other federates. As the RTI will be the center of communication among federations, RTI will use data from this service for efficient management of data exchange from source-federation to destination-federation.
- Object Management, is used to define the actual exchange of data at the level of object or interaction class. Using this service, RTI can do the book-keeping of object -instances (including its attributes) and interaction-instances.
- Ownership Management, provide RTI a way to manage the ownership of instances' attributes within federations. Using this service RTI can also guarantee that one attribute can only be owned by one instance (either object or interaction class).
- Time Management, is a mechanism for the RTI to manage the advancement of simulation time for each federate. Using this mechanism for time-stamped order message, federate can get a guarantee that it will never received an older message compare to its actual time.

- Data Distribution Management, provides the refinement of the service given by Declaration Management. It provides a mechanism to prevent the transmission and reception of irrelevant data.

A.2 HLA Object Model Template

HLA Object Models Template (OMT) is a formal definition to describe the simulated entities. SOM and FOM are defined using the common rule of OMT. SOM provides the capabilities of federates for exchanging information as a part of federation while FOM provides data exchange at the level of federation.

A.3 HLA Rules

The HLA Rules define behavior of federate and federation which comply to the HLA standard definition. It is classified in two parts, namely federate and federation rule. The definition and description of each rule is in [23].

Appendix B

CORBA Components

B.1 ORB Core

It is a CORBA component that hides all the implementations details, location, execution state and the communication mechanism of an object. To invoke an object, client only needs to know the object's reference without being able to modify it. There are different ways for a client to obtain an object reference. Client might obtain the reference by creating a new CORBA object which automatically provides the a unique reference for that new object. Other way to obtain an object reference is by requesting it via the specific service offered by other CORBA component. Examples of those services are Naming Service and Trading Service. By distributing the common task to other component, ORB can be kept as simple as possible.

B.2 OMG Interface Definition Language (OMG IDL)

In order to be able to invoke an object, a client needs to know the type of service and data that the object provides as well as the form of the object's interface. OMG IDL creates this interface independently from the object implementation so that object implemented in different programming language might be able to communicate with each other. All the data types are standardized and covers from the most basic types (such as `float`, `long`, `double`, etc) up to constructed types of data (such as `struct` and `union`). The standardize interface created by IDL is applied as a *stub* and *skeleton* for client and server respectively and used an interface to communicate with the ORB.

B.3 Language Mappings

As the IDL only creating the definition of interface for the client and server, a mapping to transform these interface specifications into a certain programming language

is needed. Several language mappings has been standardized by the OMG, including C/C++, Java, Smalltalk and Ada. An IDL compiler will generate client-stub and server-skeleton based on the information specified in IDL into some particular programming language which can be understood by the servant.

B.4 Servant

Having an IDL as a specification of an interface, a servant is a CORBA component that support the implementation of all operations provided by object interface. Servant can be considered as an instances of programming language class.

B.5 Portable Object Adapter (POA)

The client invokes an object using an object interface and the request is flowing to a servant implementing that object via ORB. The role of POA here is to dispatch a client request from the ORB to an appropriate servant which implements that object and activate that particular object. In order to be able to find the correct servant, POA might provide the service to register the servant. If the amount of CORBA object grows, POA can assign this task to other CORBA service known as a *servant manager*.

B.6 Dynamic Invocation Interface (DII)

This interface allows client to compose operation requests at run-time without requiring IDL interface-specific stubs to be linked in.

B.7 Dynamic Skeleton Interface (DSI)

At the server's side, DSI allows ORB to deliver request to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. Client is not aware if the request is passed to the servant through type-specific IDL skeletons or via the dynamic skeletons.

Appendix C

TAO ORB Configuration

To support real-time applications which has system-specific requirements for obtaining the optimal performance, TAO provides a very flexible ORB configuration in terms of *service configurator*. Service configurator is a framework that supports the creation of dynamically-configured applications. By default, the service configurator in TAO reads the ORB configuration from a file called `svc.conf`. TAO organizes its strategies and resources into three groups where each group has an aim to optimize a certain aspect explained as follows:

Resource Factory

Controls the configuration of resources used by ORB core. Some examples of options that can be configured are:

- `ORBListenEndpoints`, tells the ORB to listen for requests on the interfaces specified by endpoint(s). Endpoint can be a file or a protocol (IIOP, UIOP, SHMIOP, etc).
- `ORBInitRef`, specify an object reference for an initial service.
- `ORBSndSock`, control the sizes of hte protocol's send buffer in an attempt to maximize throughput.
- `ORBRcvSock`, control the sizes of hte protocol's receive buffer in an attempt to maximize throughput.
- `ORBConnectionCacheMax`, supplies the maximum number of connections that may be cached in the ORB.

Server Strategy Factory

Creates the configuration used by the object adapter at server-side. Several available options that can be tuned are:

- **ORBConcurrency**, specifies the concurrency strategy used by ORB to control server behavior. **Reactive** option will handle multiple clients using single thread while **thread-per-connection** assigns a new thread to service each connection.
- **ORBPOALock**, specifies the type of lock used for accessing the POA. Option **null** is used if no lock is required and option **thread** is used to specify inter-thread mutex to guarantee exclusive access.

Client Strategy Factory

Supplies the configuration that optimize object stub at client-side.

- **ORBConnectStrategy**, specifies the way clients wait while initiating connections to servers, either by blocking or non-blocking behavior.
- **ORBTransportMUXStrategy**, controls the request multiplexing strategy of the transport. **Muxed** option will multiplex pending requests on a connection and **exclusive** will allow only one request pending on a connection.

There are a lot more of the ORB options that can be configured to serve system specific requirements. In [12] the complete options for ORB configuration is covered in detail.