Design and Implementation of a HLA RTI Prototype at ONERA

Pierre Siron
ONERA-CERT
Information Modelling and Processing Department
2 av. E. Belin, BP4025
F-31055 TOULOUSE Cedex
Pierre.Siron@cert.fr

Keywords: HLA, RTI, distributed architecture, time management, security

ABSTRACT: *The increasing demands of simulations for designing aerospace vehicles need powerful tools and infrastructures allowing reusability and interoperability between distributed components. In this paper, we discuss ongoing work carried out at ONERA (Office National d'Etudes et de Recherches Aérospatiales) which aims at providing a common reusable infrastructure for distributed simulations. More precisely, we describe first the design and implementation of a HLA compliant RTI prototype, as the first step of our project. We then discuss preliminary lessons learnt in experimenting our prototype. Finally, we propose several issues for upgrading this first prototype.*

1. Introduction

The DoD has proposed HLA for the next generation of modelling and simulation projects. ONERA is strongly linked to the French Ministry of Defense (DGA more precisely). Therefore we are sharing the need for a generic distributed simulation architecture, which emphasizes the interoperability and the reusability of simulations.

ONERA is also involved in aeronautic and spatial studies and researches. After reading the HLA specifications, we are convinced that the scope of HLA goes beyond the Defense sector. For example, we have to simulate new avionic systems, which are distributed systems, involving resources sharing problems, reliability, security and real-time constraints. Is a distributed simulation of such distributed systems more suitable ? Another point is that their development implies many companies. The cooperation of distributed simulations is then a key problem. To answer these questions, our position is not to stay and wait for commercial products but to start a first step: the design and the implementation of a RTI (RunTime Infrastructure) prototype at ONERA.

2. Objectives

The main objectives of our initiative are the following:

- First of all, to get a better understanding of the HLA architecture from several points of view. For example, what kind of difficulties have to be overcome in designing and implementing the RTI from its specifications [1]. Another important issue is to evaluate the underlying programming methodology of HLA in order to provide relevant support for the potential users inside ONERA.
- Secondly, to initialize new researches, suggested by performances issues and/or new simulation paradigms. We could here apply our skills in distributed systems and in computer security.
- Finally, to provide a HLA architecture with security properties, which is an important issue in both the defense and the civil domains, as soon as several companies have to cooperate on the same project.

3. Prototype

3.1 Schedule

A small team was set up to produce a RTI system. The project started by the end of 1996. A first version was available in September 1997. A second one was achieved in May 1998, which is more complete and includes some optimizations.

The objective is a rapid prototyping, therefore the design and the implementation must be incremental. We have selected a reduced set of HLA services, which are enough to build significative applications, without data distribution management and without ownership management. This looks like the Familiarization version of the RTI [2].

3.2 Assumptions

An object design and the C++ language were used for the RTI code, since the language is not a constraint for the development of the federates. We used standard protocols, such as TCP/IP and UDP, for the communications between the different components. We did not choose a middleware like CORBA for the same reasons as those described in [3] (performances, inappropriate client-server paradigm).

3.3 A prototype architecture

3.3.1. Various components

Our prototype has some original characteristics, in particular it is built around an architecture of communicating processes. The RTI is a distributed system involving two processes, a local one (RTIA) and a global one (RTIG), and a library (libRTI) linked with each federate. The RTI architecture is depicted by Figure 1.

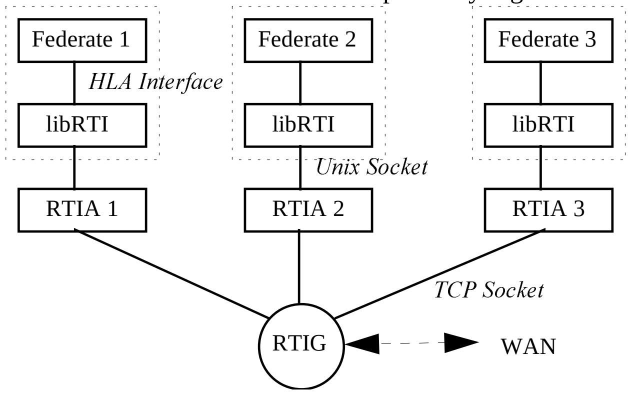

Figure 1: RTI architecture

The first step of our development concerns a local network (Ethernet) of Unix (Solaris) workstations (Sun). One federate and its RTIA process are located on the same computer. The RTIP runs on another workstation (the most powerful).

3.3.2. The RTIA

Each federate process interacts locally with a RTI Ambassador process (RTIA) through a Unix-domain socket. The RTIA processes exchange messages over the network, in particular with the RTIG process, via TCP (and UDP) sockets, in order to run the various distributed algorithms associated with the RTI services.

A specific role of the RTIA is to satisfy immediately some federate requests, while other requests require message sending. The RTIA manages memory allocation for the messages files and is always listening to both the federate and the network (the RTIG). It is never blocked because the required computation time is reduced.
Various objects are summed up in the figure 2. To simplify, there is one object for each implemented service family of the RTI: federation management, declaration management, object management and time management. The time management object uses a LBTS management object (lower bound on time stamp [4]. The root object reflects the HLA object classes, the interaction classes (the SOM) and the instances. Files management object contains files for the various types of messages: Receive Order or Time Stamp Order. The communication management object is an abstract class which hides the used protocols and the needed optimizations.

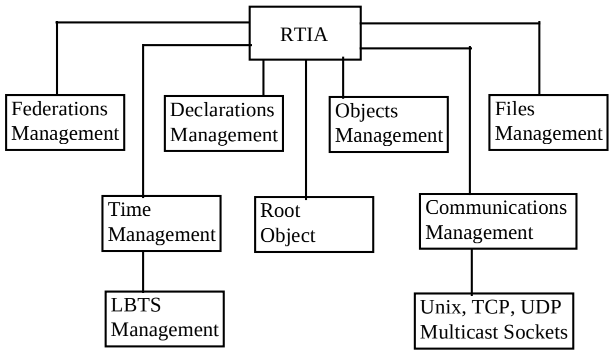

Figure 2: RTIA structure

3.3.3. The RTIG

The RTI Gateway (RTIG) is a centralization point in the architecture. Until now, its role has been to simplify the implementation of some services. It manages the creation and destruction of federation executions and the publish-subscribe data. It plays a key role in message broadcasting which has been implemented by a multicast approach. Therefore, when a message is received from a given RTIA, the RTIG delivers it to the corresponding RTIA, avoiding a true multicasting.

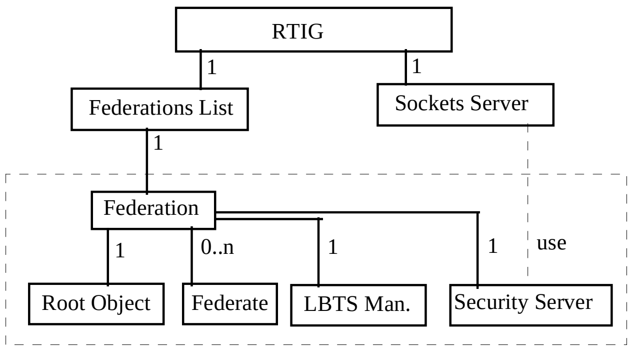

Figure 3: RTIG structure

Figure 3 shows various objects. The Socket Server object manages a list of opened sockets (the current connections) and the communications.

3.3.4. The libRTI library

A service call is simply transformed into the building of a message (which includes a type and the input parameters), its sending to the associated RTIA, the waiting from the response of the RTIA (usually an acknowledgment), and the presentation of the output parameters.

A tick primitive is added, allowing the execution of the RTI initiated services (user code associated to TimeAdvanceGrant, ReflectAttributeValues, etc).

3.3.5. Data-transfer scenario

Figure 4 illustrates the exchanges of message involved by an UpdateAttributeValues (this is like a data transfer). The message file is not represented in the right-side RTIA, nor the delivery condition which depends on the time management [4]. UAV is the acronym of UpdateAttributeValues, and RAV refers to ReflectAttributeValues.

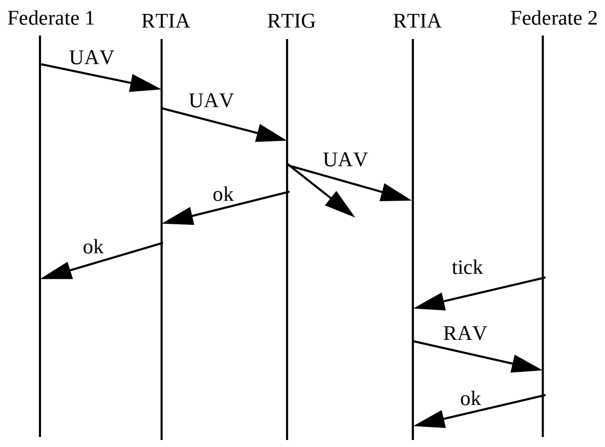

Figure 4: Data-transfer scenario

4. Applications

4.1. Test applications

The first category of applications is test (or toy) applications. These applications use all the RTI services which are implemented, they are also significative for the number of exchanged messages and the consumed bandwidth.

The first application is a billiard game. There is one main object class, the ball with position attributes. The federation is composed of any number of federates, each federate modellizing and simulating only one ball instance. It publishes and subscribes to the position attributes, so that each computer could graphically represent the current situation. The collisions are simulated by interactions. This is a time-coordinated example, with an initial synchronization point.

A sub-class of ball, colored ball, adds a color parameter. This example helps us to test the inheritance mechanisms.

The second example is a tennis game. This is a partially a reuse of the ball example. We add a racket class which is instancied by two new federates. A new interaction is also defined for the collision between a racket and a ball. A monitor federate subscribes to the position attributes of the objects and visualizes the court. The monitor is time constrained but also time regulating for a more realistic result.

4.2. Air defense application

This example comes from the Escadre toolkit. Escadre is a simulation support environment developed by DGA/CAD (Centre d'Analyse de Défense). It provides an OO methodology and a tool box for developing and running constructive simulations [5]. A typical application [6] is a fight simulation of aircraft attacking air defense units. Patrols of aircrafts are equipped with anti-radar missiles. Air defense units are composed of a command post supervising multifunction radar devices and surface-to-air missiles ramps.

We have chosen this problem because it is more significative and a little more complex. The object classes are Aircraft, Radar and Missile which are derived from a Target class. The interactions are Radar Emission, Destruction, etc. The federates are associated with the Aircrafts, the Defense Units (which both simulate the Missiles) and a Monitor like in the previous example.

5. Conclusion

5.1. Difficulties and optimizations

The distributed algorithms for the implemented RTI services were well-known (the time management for example). There is only some recursive algorithms to take into account class inheritance.

As a consequence the RTIA and the RTIG processes, and in the same way the libRTI, are relatively simple. But the communication structure is still a open problem. An optimization consists in decreasing the number of exchanged messages and the amount of data per message. This implies a precise definition of the message structure and the use of variable length. We did not have a good socket encapsulation (only a C library, not a C++ class) and some optimizations of the TCP ptotocol and of the RTI prototype were conflicting.

To manage federations, publish-subscribe data, message multicast, a regret is not to have at disposal a efficient tool for group management and communication system. For example the Horus system could also facilitate the implementation of the causal order option [7].

To conclude, we have now an operational RTI prototype with significative applications. This strengthens our hope in HLA, and in particular the definition of a minimal set of services to perform distributed simulations.

5.2. On-going work

An on-going work is to tackle the security problem of the distributed simulations. Because we have the code of a RTI, because we control the overall architecture, we can add some security mechanisms. This will be detailed in [8].

5.3. Future trends

A first perspective is related to the applications. We are proceeding to a selection of significative applications at ONERA.

We do not want to produce a complete or commercial RTI, our interest is rather in difficult points which necessitate still some research, for example the Time Management, and specially the zero-lookahead problem and the causal order option. This last option could be useful to improve performances, but we have to study its impact on the applications.

Another issue is the multi-resolution problem. For example, in the air defense application, the Defense Unit is the aggregation of a command post, radars and missile ramps. What can we do with HLA ? How can we extend it ?

Work remains to be done at the communication level. We plan to conduct some experiences with an ATM network. From a software point of view, the connection of distributed simulations by a WAN could be made by the connection of various RTIG processes. We could then evolve a commonly acceptable definition of a gateway.

6. References

[1] Department of Defense, "High Level Architecture Interface Specification, Version 1.3 Draft 7", January 1998.

[2] R. A. Briggs, R. M. Weatherly, R. Richardson, J. Olszewski, T. Stark, "RTI F.0 integrated product team", Proceedings of the Spring Simulation Interoperability Workshop, 1997.

[3] M. Furuichi, M. Mizuno, H. Miyata, M. Miyazawa, S. Matsumoto, K. Aoyama, "Design and implementation of experimental HLA-RTI without employing CORBA", Proceedings of the 15th DIS Workshop, 1996.

[4] Department of Defense, "HLA Time Management Design Document", Version 1.0, August 1996.

[5] C. Sautereau, "Escadre V4", CAD report, April 1995.

[6] J-L. Igarza, C. Sautereau, P. Annic, E. Berry, D. Canazzi, "Development of a HLA compliant version of the French Escadre simulation support environment: lessons learned and perspectives", Proceedings of the Spring Simulation Interoperability Workshop, 1998.

[7] R. van Renesse, K. Birman, S. Maffeis, "Horus: a Flexible Group Communication System", Communications of the ACM, April 1996.

[8] P. Bieber, J. Cazin, P. Siron, "Security Extensions to ONERA HLA RTI Prototype", submitted to the Fall Simulation Interoperability Workshop, 1998.

Author Biography

PIERRE SIRON was graduate from a French engineer school of computer science in 1980, and received his doctorate in 1984. Then he is a Research Engineer at ONERA and he works in parallel and distributed systems and computer security. He is a member of the Design and Validation of Computer Systems research unit.