Using HLA to provide a Collaborative Multi Agent Virtual Prototyping platform

Valéry Raulet
Vincent Rodin
Alexis Nédélec
Software Engineering Laboratory (LI^2)
École National d'Ingénieurs de Brest (ENIB)
Parvis Blaise Pascal
Brest, 29200, France
(033) 2 98 05 66 31
raulet@enib.fr, rodin@enib.fr, nedelec@enib.fr

Keywords:
High Level Architecture, Multi Agent System, Collaborative Prototyping,
Distributed Virtual Environment, Virtual Reality

ABSTRACT : *In a previous paper[6], we presented how we provided a dynamic object model with the High Level Architecture (HLA). In this paper, we present how we incorporate the HLA framework into oRisDis, our multi agent oriented simulation platform, and the dynamic object model. Our platform adds an additional Run-Time Infrastructure, called oRisRTI, which provides a framework between agents and the HLA RTI. While the HLA RTI provides necessary services for data exchanges between platforms, the oRisRTI provides services specific to multi agent collaboration and higher level services (dead-reckoning, static and dynamic state update management, ...).*

We will also emphasize difficulties for integrating real/ghost agent model management into the HLA exchanges and how benefits could be gained into adding further functionalities to the HLA architecture.

1. Introduction

Originally developed for single user applications, our platform called oRis is being extended to provide collaborative prototyping. Lots of solutions exist for allowing sharing between hosts (PVM, MPI, Linda, CORBA, ...). We made the choice of the High Level Architecture (HLA) because of its design and its standardization for distributed simulations.

Indeed, HLA uses an object-oriented model. Our platform is multi agent oriented and this coupling can be easily made according to object and agent similarities.

This paper describes how we realized the interconnection between our multi agent platform and the HLA architecture. Firstly, we briefly present our platform, then needs for communicative platform are described and solutions retained are shown.

2. The oRisDis platform

oRisDis, our distributed platform, consists of three main components. First, oRis is the core architecture which provides a multi agent language and a simulation engine for agents described in this language. The second component, ARéVi, is a 3D rendering API plugged around the oRis core. It provides many capabilities for rendering and external interactions. The last one, oRisDis is our work in progress to provide distributed facilities to the oRis/ARéVi environment. Based on HLA, it provides data sharing. This data sharing is done the most transparently as possible to avoid bothering the designer with distribution problems.

2.1 oRis environment

oRis[1][2], the core component, is a Multi Agent System (MAS) developed in our laboratory. Built for many platforms (IA32 Linux, PPC Linux, SGI Irix and Windows), oRis provides an agent language and a simulation platform.

oRis provides a original prototyping tool. Indeed, instead of designing a prototype in phases between off-line and on-line design, oRis provides enough flexibility to fully modify prototype during simulation.

2.1.1 Prototyping

Prototyping consists in designing a model as expected. Classical prototyping is a round trip between off-line design and on-line tests. This way of acting tends to disappear in favor of complete numerical design. We want to go a step further by allowing its designer to modify its model while testing. Figure 1 shows interactive prototyping where adjustment are now done on-line.

[1] *http://www.enib.fr/~harrouet*

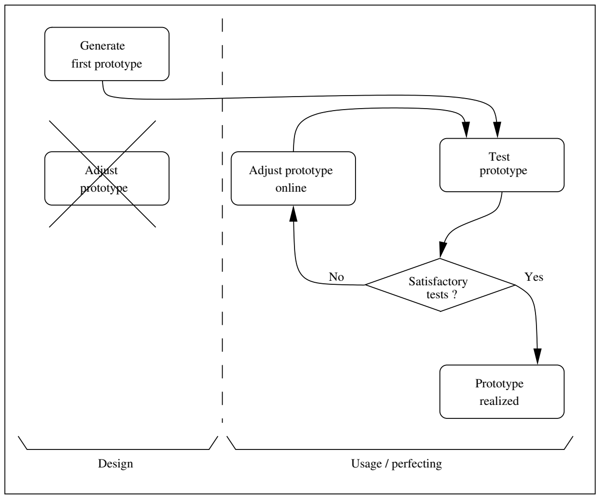

Figure 1. Interactive prototyping approach (from [2])

2.1.2 An agent language

Based on a syntax similar to C++ and Java, oRis is a language very close to object oriented languages with few more features.

Features provided by this language are agent behavior, the `main()` module, agent communication capabilities, and dynamic behavior evolving.

In the `main()` module, we can describe the agent's behavior. At the simulation startup, this module is called and represent the agent's entry point. Then, this module is periodically called, boundlessly, by the simulation engine.

Agent communication capabilities allows agents to share data. This can be simple data such as entity state or more complex information described in ACL[2] language. ACL is a formalized way for communicating between agents. FIPA[3] or KQML[4] are some of these ACL[4].

Dynamic evolving is a specificity of our platform. Indeed, during execution, agents can evolve in a way that was unpredicted by its designer. The whole language is available at runtime and is interpreted. We call this a *dynamic language*.

2.1.3 A simulation engine

Designed for simulating systems behaviors, the platform provides a simulation engine for making live agents. A fine tuned scheduler has been designed to avoid bias and thus each agent lives with fairness to its neighbor.

The aim of the engine is to simulate each agent behavior without privileging one of them and to provide complete access to the agent code and the oRis language. This allows an user to modify the application while running.

2.1.4 ARéVi Toolkit

Based on oRis core component, ARéVi[3][5], our Virtual Reality Toolkit provides a 3D graphical environment. It allows entity representation in three dimensions and user interaction with adapted devices.

It can be used as a stand-alone tool for rapid prototyping or can be embedded into an existing system. Based on notions of entities, scene (group of entities) and viewers, it handles various features such as animation, level of details, lightning, collision detection…

Interaction is provided through usual or VR peripherals like 3D mice, force feedback joystick, flock of birds, Phantom and so on.

2.2 oRisDis Multiuser Platform

oRisDis is made of two parts: a plug-in C++ module for oRis and some oRis code. The module provides bindings between oRis language and the C++ HLA API.

The oRis code, called *oRisRTI*, is another RTI designed specifically for our platform. Agents (real or ghost) living onto the platform can interact with this *oRisRTI*. This latter is in charge of translating agent calls into HLA calls. Conversely, it translates HLA calls into oRis agent module calls. This is shown on figure 2 and is detailed in the next section.

3. Distributed agent platform needs

HLA is designed for providing a useful generic API for distributed simulations. It provides different services which can be used for many purpose. Despite its genericity, we have already shown that HLA is not sufficiently opened for every uses[6].

The oRisRTI provides more specific services, based on HLA generic ones, to our platform.

3.1 Ghost/Real interaction paradigm

Our platform is based on a real/ghost interaction paradigm. A real entity is the one living on its creator platform while ghost entities are representative from this real entity. Real

[2]ACL : Agent Communication Language
[3]FIPA : Foundation for Intelligent Physical Agents
[4]KQML : Knowledge Query and Manipulation Language
[5]ARéVi, ***A**telier de **Ré**alité **Vi**rtuelle*

entity does support a full realistic behavior which can be computer driven or user driven. On the over hand, the ghosts entities does embed a “lighter” behavior which mimics real behavior.

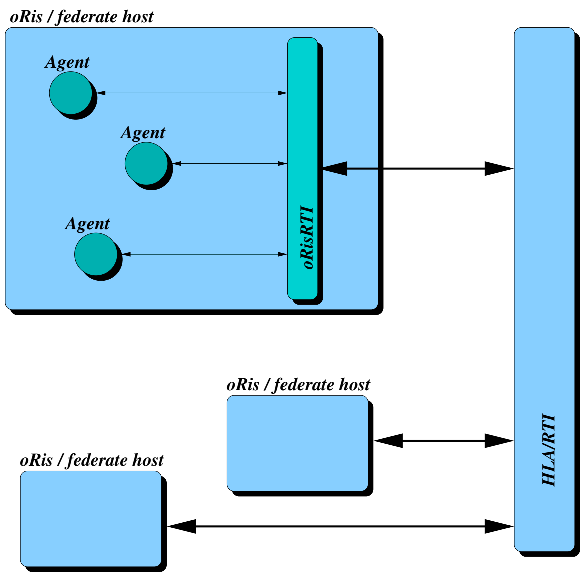

Figure 2. Double RTI infrastructure

Dead reckoning[7] is an example of such a behavior. The ghost behavior is to reflect more simply its real movement. This is done by exchanging location, velocity and acceleration. Ghost tries to mimic its real entity, graphically, by moving on a near path. On the other hand, real entity can have a very complex behavior resulting in its move. Reasoning is done once but location is the same on different hosts.

More than that, we also want to be able to interact with ghosts. While they do embed real behavior of its real entity, interactions must be reflected to its real entity which is in charge of taking the correct action.

This two way action can be problematic in case of a dynamic entity state update. This is explained in next subsections.

3.1.1 Bidirectional vs unidirectional interaction

In interactive distributed simulations, environment state is maintained by exchanging entity state attributes which reflect entity representation (principally visual and auditive). Interaction between entities – or between users – is done by exchanging interactions instances (in HLA terms). In distributed simulations, this is not problematic since entity is “owned” by one user. You can’t move its entity without asking him to do so !

In our prototyping platform, an entity is not specifically owned by its creator. Another user must be able to interact with it. Interactions can be really bothering due to latency[5]. Ghost agents can be modified locally while referring its action to its real entity. This is not problematic for single shot action but is more difficult to manage for dynamic state modification, as inserting new properties to agents.

3.1.2 Entity state update frequency

Entity state attributes doesn’t evolve identically. Some of them have a static state evolving – one shot – while others have a really dynamic state evolving (entity movement, heating color, . . .). This can be particularly embarrassing for our platform since interactions can occur in a two way fashion.

A typical interaction between a ghost and its real entity and between real and its ghosts representation is displayed on figure 3.

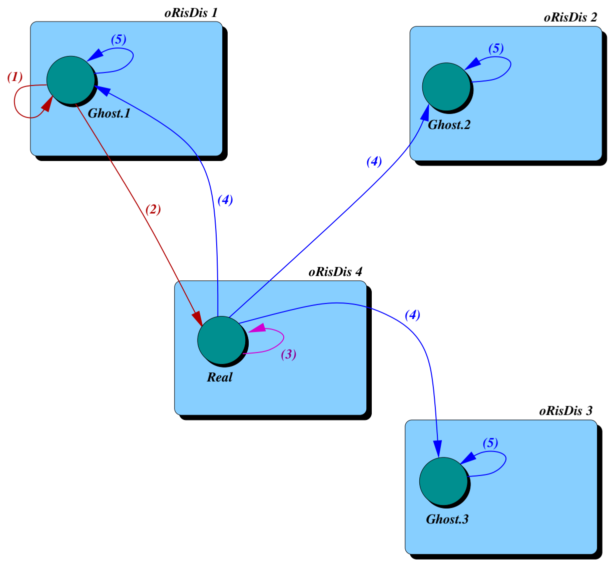

Figure 3. Real and ghosts interactions

A dynamic state interaction occurring on a ghost is typically done when someone uses a manipulator to interact and move a ghost entity. For example, the ghost entity has been designed to send an interaction to its real when a threshold (5 inches for example) is over-stepped. If a user is moving this ghost entity 40 inches farther, eight interactions are going to be sent to inform its real entity. But during this move, real entity is going to reflect interaction received. This results in a back and forth moving. Latency is going to introduce delays between action being taken and action been taken reflected by real. Figure 4 shows such a problem.

In this example, a ghost is sending periodically updates to its real associated entity. Each time this real receives a message, it decides to update itself and to send an update to its ghost entities. But latency is such that ghost has enough time to send many messages before receiving any update from real entity.

On the bottom on figure 4, we display what may result in such a situation. (1) represents the initial behavior of the interactor and (2) represent the resultant behavior caused by real updates.

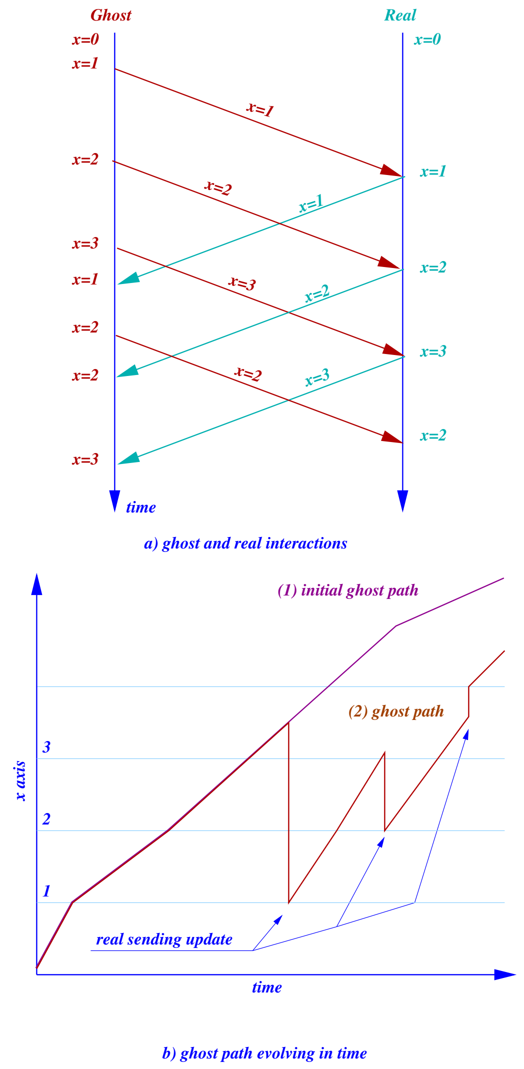

Figure 4. Dynamic state attribute evolving with real entity updates

Solution When a ghost modifies a dynamic attribute state, two types of updates can interfere.

First one is caused by its own updates sent back by real entity. This problem is simply solved by transmitting its updates with an identifier specifying who is the originator of this update. When a ghost receives an update originating from it, it simply discards this update.

Second one occurs when two (or more) users try to interact on ghost entities representing the same real one. We choosed a simple solution to this problem. When real entity receives updates from different ghosts, an average is computed and the shift position is sent to all ghosts. In this case, ghosts displacement is hindered by other users actions.

3.2 Communication between agents

Participating agents can be intelligent agents[8]. This means that communication can occur between agents. This is modeled by point to point or broadcast communication between agents. Needs are different for each one.

In unicast communication, the destination agent is known. This is not problematic since ghost representing entity can forward the message to its real entity.

In broadcast communication, the problem is that emitter agent doesn't know who has to receive the message. To answer this point, we implemented a specific agent – in the oRisRTI – which is sensitive to every messages that can be broadcasted. Figure 5 display agent organization for messages exchange.

On top of the figure, an agent broadcast a message (agent 1) to all the agent sensitive to this message. Real agent present on this platform receives the message (shown with a dashed circle around the agent). Ghost agents are not sensitive to any broadcast message but the specific agent (agent 2) – in the oRisRTI – also receives this message. Then, this message is sent to the HLA RTI through an interaction (`Send Interaction With Region`).

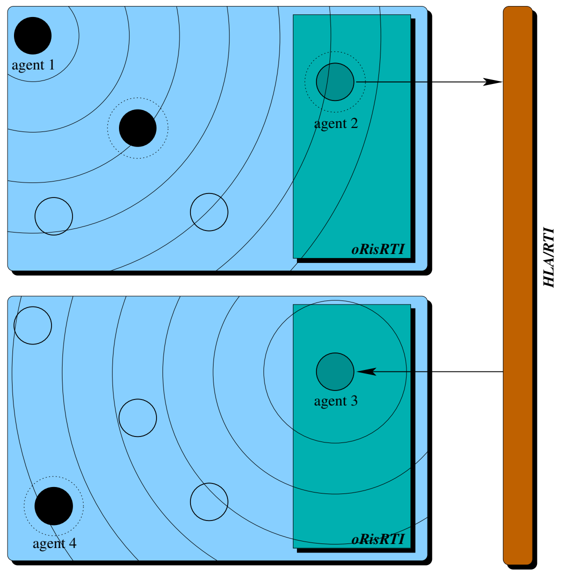

Figure 5. Infrastructure for agent communication exchange

On the other side, an agent (agent 3) receives a specific message originating from the HLA RTI (`Receive Interaction`). It extracts data and recreates the message. This message is then broadcasted by a local agent onto this platform (agent 4 receives this message).

Solution details A specific interaction class has been designed for messages broadcasted by agents. Each oRisRTI subscribes to this interaction class and associates a region with it (`Subscribe Interaction Class With Region`). A specific *routing space* exists for this interaction.

On each platform, when an agent become sensitive or insensitive to a type of message, it informs the oRisRRI agent (agent 2 or 3 on figure 5). This latter uses this information to create, modify, or remove an extent for the region associated to this interaction. Subscription is modified accordingly.

Extent associated with a message is maintained into a table (message name, extent). This information has to be agreed between each federate. Since new messages can be created during execution, this agreement is done by sending an interaction with a specific region that is used for adding a new entry into the table (maintenance interaction). By acting this way, we hope that two platforms will not send the same message creation simultaneously.

This solution avoids to send interactions to federate that are not interested by this agent's message type.

3.3 Dynamic evolving of agents

The oRis platform is able to add, modify or remove an agent while executing. This feature is so important that it must remain available in the collaborative one. This mean that the object model can be altered while needed. HLA provides a fixed unmutable object model.

Our retained solution was presented in a previous paper[6]. The oRisRTI is in charge of encapsulating the oRis object model into the HLA object model. Initial oRis and HLA object model are identical. But when new attributes or classes are added to the oRis object model, this is hidden to HLA by using a generic attribute and a specific protocol.

The generic attribute is associated with the new oRis added attribute. A specific protocol is used for platform agreement. When a new attribute , for example *foo*, is added to an oRis instance, each platform has to agree that it is related to the same attribute. This encapsulation realizes association between the attribute name (only relevant for oRisDis) and the attribute handle `1` from instance handle `23` from the HLA.

This solution keeps some services available but we are examinating a new solution which require to modify the HLA API to introduce method to allow object model modifications. This is currently being done on the *CERTI RTI*[6][1].

[6]http://www.cert.fr/CERTI/

4. Conclusion

HLA is really an interesting infrastructure for a lot of projects. It provides a sufficiently generic API for targeting different applications. Our project shows how HLA can adapt to different needs as presented in this paper.

HLA is a first layer for us and more functionalities have to be added to provide a more useful API. Each domain may have to provide a more specific layer adapted to their needs to improve interoperability.

But, as always an API is never sufficiently open for every usage. This is also the case with our platform since static object model is not enough. We are working on adding new API methods for allowing HLA to modify its object model during execution.

Acknowledgements

We would like to thank the CERTI team who released their API to the GPL license. This allows small teams as us to use complex API in research projects.

References

[1] B. BRÉHOLÉE AND P. SIRON, *CERTI: Evolutions of the ONERA RTI Prototype*, in Fall Simulation Interoperability Workshop (02F-SIW-018), Orlando, USA, September 8–13 2002.

[2] F. HARROUET, *oRis : in immersion through the language for virtual prototyping based on multi agents (in French)*, PhD thesis, Université de Bretagne Occidentale, Equipe d'accueil 2215, Laboratoire d'Informatique Industrielle, Décembre 2000.

[3] F. HARROUET, P. REIGNIER, AND J. TISSEAU, *Multiagent systems and virtual reality for interactive protot yping*, vol. 3, ISAS'99, Orlando (USA), July 31 - August 4 1999, pp. 50–57.

[4] A. NÉDÉLEC, P. REIGNIER, AND V. RODIN, *Collaborative Prototyping in Distributed Virtual Reality Using Agent Communication Language*, in IEEE SMC'2000, Nashville, USA, Octobre 2000.

[5] V. RAULET, A. NÉDÉLEC, AND V. RODIN, *Coupling HLA with a MAS based DVR environment*, in IASTED International Conference – Applied Informatics, Innsbruck, Austria, February 18–21 2002, pp. 239–242.

[6] V. RAULET, V. RODIN, AND A. NÉDÉLEC, *orisdis: using hla and dynamic features of oris multiagent platform for cooperative prototyping in virtual*

environments, in European Simulation Interoperability Workshop 2002 (02E-SIW-022), London, UK, June 2002, SISO.

[7] S. K. SINGHAL, *Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments*, PhD thesis, Thesis, Department of Computer Science, Stanford University, Août 1996.

[8] M. WOOLDRIDGE, *Multiagent Systems – A modern Approach to Distributed Artificial Intelligence*, Massachusetts Institute of Technology, 1999, ch. Intelligent Agents, pp. 27–77.

Author Biographies

VALÉRY RAULET is a PhD student in Computer Science at the École Nationale d'Ingénieurs de Brest (France). His work aims at providing a toolbox for building distributed and collaborative applications.

VINCENT RODIN is born on February 28 1966. Lecturer at the École Nationale d'Ingénieurs de Brest (France), he's currently working on image processing, virtual reality and computer simulation of biologic processes.

ALEXIS NÉDÉLEC is born on June 03 1961. Lecturer at the École Nationale d'Ingénieurs de Brest (France), he's currently working on Agent Communication Language (ACL) in Multi Agent System for collaborative applications development in virtual reality.