oRisDis : using HLA and dynamic features of oRis multi agent platform for cooperative prototyping in virtual environments

Valéry Raulet
Vincent Rodin
Alexis Nédélec
Software Engineering Laboratory (LI^2)
ENIB
Parvis Blaise Pascal
Brest, 29200, France
email: raulet@enib.fr, rodin@enib.fr, nedelec@enib.fr

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

ABSTRACT : HLA is a framework providing functionalities for distributed simulations. It allows creation and destruction of federations, synchronisations, sharing objects and so on, but a way to create objects dynamically is lacking.

Our aim is to develop a distributed platform for Collaborative Virtual Environment called oRisDis. This environment will be based on oRis, an agent-oriented language and simulation engine which allows agents activity management. Dynamic prototyping is provided with oRis by *immersion through the language*. This means that a user can modify or add agents while running. Of course, such a feature has to be kept in oRisDis.

In this paper, we present our architecture providing a framework for Distributed Collaborative applications. Its main specificity is the dynamic object management service.

1. Introduction

For many years, the Department of Defense (DoD) strives to standardize network communications between simulations. DoD is certainly the most important client in simulation softwares and its main goal is to reduce cost in developments. The first normalization was the Distributed Interactive Simulation protocol (DIS) which implemented a number of Protocol Data Unit (PDU). Many distributed platforms used this protocol to implement communications between users but its main disadvantage was its orientation towards military simulations. Some of them implemented extensions to provide more general PDU.

On the other side, Aggregate Level Simulation Protocol (ALSP) was available. Aimed at providing a support for parallel discrete event simulations, it supports interoperability[3]. These two architecture are now merged into a single one called the High Level Architecture (HLA). The HLA core is the Run-Time Infrastructure which can be seen as a middleware managing messages exchange between different users. This RTI provides six types of services (federation, object, declaration, data distribution, ownership and time management) which can be divided into two groups : a part which manages federation and a part managing the objects' attributes.

Management of objects information first needs to be described in a specific file '.fed' which describes the object model of the simulation. To interoperate, each federate needs to use the same object model which provides them knowledge of what objects will be available during federation execution. So each federate will be able to create or discover these objects, send or receive updates from objects' attributes.

Such an architecture is adequate for simulations since it simulates a scenario. It means all informations exchanged during execution are known before simulation begins. But this way of acting does not scale to every distributed environments. "Imagine a net-VE participant bringing a new, previously unseen object into the environment and being able to see, hear, and interact with instantly and meaningfully" as stated [9]. This is the Holy Grail that every net-VE would like to develop.

In section 2., we present our platform based on two components called oRis and ARéVi which provides a multi agent based prototyping tool for a single user. This section will highlight specific features provided by this environment. Next section 3. will discuss about dynamic incompatibilities between oRisDis and

HLA. Then section 4. will explain how we circumvent this limitation for providing full dynamicity into our distributed platform.

2. oRis/ARéVi platform

The core of our architecture, oRis[4], is a Multi Agent System (MAS) developed in our laboratory. Available on many platforms (i386 Linux, PPC Linux, SGI Irix and Windows), it allows the execution of applications developed with agents. oRis is an agent language and also a simulation platform.

oRis is an object oriented language. This means that object oriented programming style is used. Thus, entities are described by attributes, methods and classes with multiple inheritance. Its syntax is close to Java and C++. Moreover, oRis is also an agent oriented language. This is done by providing a special method : *void main(void)*. Available in each agent, this method describes the agent's behavior. It's the entry point for agent execution. The scheduler infinitely reexecutes this method.

By being interpreted, oRis doesn't attain performance from compiled languages such as Java or C++. But low level code has been optimized through a deep coupling with the C++ language. Native classes in oRis have been coded in C++ to speed up the application execution. But oRis takes advantage of its interpretation by providing dynamicity. Dynamicity enable modifications into the application while executing.

This mean that during execution, availability is given to modify the application behavior to the desired objective. This functionality is really important in prototyping since it allows modifications without recompiling or restarting the application.

Extending the oRis core, ARéVi[5] which stands for *Atelier de Réalité Virtuelle* or Virtual Reality Toolkit is a graphical environment which allow representation of entities into 3D. This toolkit can be used as a stand alone tool for rapid prototyping or can be embedded into an existing system. Based on the notions of entities, scenes (group of entities) and viewers, ARéVi handles various features such as animations, level of details, lightning or collision detection. Interactions is provided through usual peripherals or specialized peripherals like 3D mice, force feedback joysticks, flock of birds and so on.

3. HLA in DVR

The HLA is an architecture developed to provide a common architecture for Modeling and Simulation (M&S). This architecture facilitate reuse of individual federates and interoperability between federates.

When planning an execution, the designer needs to define a Federation Object Model (FOM) which describes the set of object classes and the set of object interactions that will be used during execution. Moreover, the attributes and parameters of these classes are also described. The designer prepares a "scenario" by describing what classes are going to be used for representating entities.

For simulations, this way of acting is enough since no code or behavior changes aren't going to occur. Everything has been prepared before execution : what interactions are available and how do they react to these interactions.

Except the shared database synchronized through RTI services, a federate application acts like any parameterized applications. For example :

```
if (receivedInteraction = ''MissileImpact'')
  then
  if ( receivedInteraction.strength >= 10 )
     then
      isDestroyed = true
     else
      damage += receivedInteraction.strength
     endif
endif
```

The Run-Time Infrastructure (RTI) is initialized by the Federation Execution Data (FED) file but after starting, no modifications can occur except attribute transportation scheme.

In Distributed Virtual Reality (DVR) or Collaborative Virtual Prototyping applications, needs for code or entity modification is required. We must be able to add new entities coming from an external file or a network link. We need to :

- create new classes not "conceived" before starting,
- modify existing modules in a class,
- modify existing modules in an instance.

These requirements lead us to consider the FED file as an initialization file that describes only a part of exchanged information.

3.1 Solutions with HLA

If we want to use the HLA architecture and provide dynamicity, we need to encapsulate HLA into a broaden mechanism. Two solutions are thinkable :

1. Use HLA when objects are known before runtime and use another communication interface for other objects,
2. Translate objects designed at runtime into known objects and use a specific protocol.

The first one is not really interesting since we need to redefine every RTI services for objects dynamically added. The way of acting would be thinkable for very specific needs like using a specialized network – with high bandwidth or very low latency – on only a small subset.

The second one needs to define a protocol between federates to define how object instance attributes are going to be exchanged and how to associate the code with those attributes. This last solution is the one retained by us.

To attain this dynamicity, we must strive to find a solution using HLA feasibilities while maintaining the most services usable. Indeed, we could simply transmit each update coming from new unknown object attributes through RTI interactions but this way of acting wouldn't keep HLA services available for these attributes.

Our solution keep part of services available by taking advantage of two points available in the HLA design :

1. No type has to be specified to an attribute. If we need to change data format, from real to string or any other conversion, HLA doesn't take care of this since every attribute exchanges are format independent. The designer is responsible to care of what is exchanged.
2. No limitations for object or interaction instances. If an object needs to be created hundred times, HLA does support it.

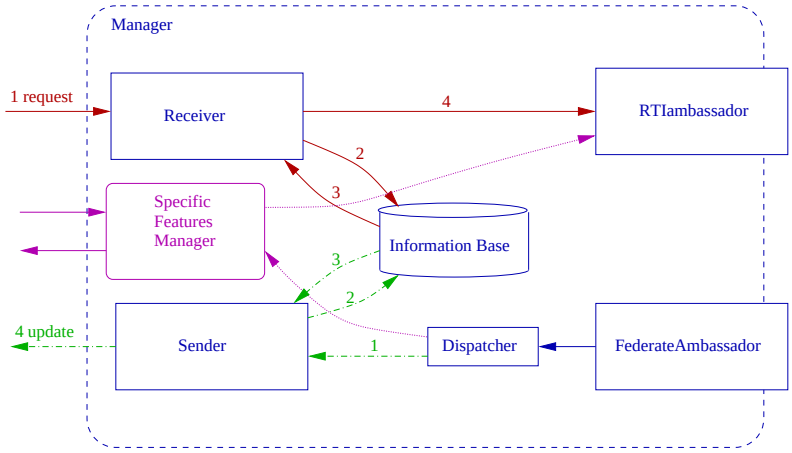

Figure 1. Manager's internal components

4. oRisDis architecture

The main goal of the oRisDis architecture is to provide, to every users participating into the execution, a common representation of the simulated world. On one side, we have the oRis application built with agents. This means state variables and behaviour are embedded into the agent himself. Moreover, the application allow users to interact with the environment by using interaction devices. On the other side, we have the simulation executive which is in charge of managing information exchange. Exchanges between the application and the others simulation executives is done through sending, receiving and delivering the events.

To enable sharing of the common world, requirements are made on agents. Firstly, agents have to be able to send their changes to the simulation executive. Secondly, simulation executive have to reflect changes coming from foreign agents. To meet these goals, we use the real/ghosts model[2]. In this model, real represent the local entities which have full capabilities (behaviour). These entities code is the one that the designer modeled. Conversely, the ghost represent the real in a simplified way. Based on state information exchanged, the ghost tries to meet the real entity behaviour (visual aspects, hearing, behavioural, ...).

More than simply reflecting the real behaviour, the ghost has to be interactable. It means that if a user interacts on a ghost entity, the entity have to transmit these interactions to the real one. A full path from a

ghost is shown on figure 2[7]. The steps are as follows :

1. A user on session 2 is interacting on its local ghost entity called Ghost1.
2. This interaction made modifications that need to be reported to the real entity. So, modifications are transmitted from the ghost to the simulation executive and sent to the real simulation executive.
3. On receipt, the simulation executive transmit information to the real entity. This entity applies the modifications locally.
4. As internal modifications have been made, the real entity send updates to the simulation executive. Network transmission occurs.
5. On receipt, ghosts (Ghost1 and Ghost2) are locally updated.

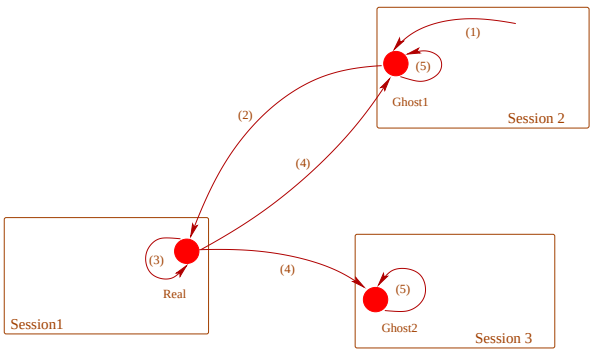

Figure 2. Real and ghosts interactions

In HLA, these information exchanges between federates or sessions are made with the use of object class instances or interaction class instances. The shared internal state of an agent is represented by an HLA object class instance. So, every update on the state of an agent is transmitted to the other federates via the Update Attribute Values service. As an attribute can be owned by only one federate at a time, a ghost can only talk to its real via the Send Interaction service.

All these exchanges are made through HLA. To simplify the prototyping tools development, exchanges between the ambassadors (RTIambassador and FederateAmbassador) are made through a more adapted interface called the Manager Executive.

4.1 oRisDis Manager Executive

Being the interface between the application et the simulation executive, the manager provides a set of higher level services. This element manages every exchanges made with the different sessions into the execution. It can be split into three parts. The first one consists of the HLA ambassadors. This is required for exchanging information between the federation executive and the RTI. The second part is the dynamic object management service (based on Sender, Receiver and Information Base). It provides the same functionnalities as the RTI object management service except that instance level dynamicity is taken into account. Last, the third part extend the manager capabilities by providing entry points (Dispatcher and Specific Features Manager).

4.2 Dynamic Object Management

Main point in the architecture, the dynamic object management part extend the HLA object management service by allowing dynamic (while executing) adding of new classes or new attributes not previously declared in the object model. One solution envisaged by Naud [6] is to restart the RTI without stopping the federates. We don't think that this solution is viable since restarting the RTI may take a while for being ready. Instead of this, we are using an intermediate for hiding the real implementation of the object model.

This is achieved by :

1. creating a generic class with an associated attribute which will "carry" the value associated to the new added attribute,
2. having a database which will keep information equivalence between oRis and HLA,
3. adding components between the application and the ambassadors. It will act as an intermediary.
4. providing a protocol to synchronise the local database disseminated accross the different federates.

4.2.1 A class for a dynamic attribute

The object model needs to be defined before execution into the Federation Execution Data (FED) file. While running, no modifications can occur. Thus, only existing attributes and classes can be used. To prevent dynamic adding, a particular class compound with only one attribute is defined :

```
(class ObjectRoot
  ...
  (class Dynamic
    (attribute Attribute reliable timestamp)
  )
)
```

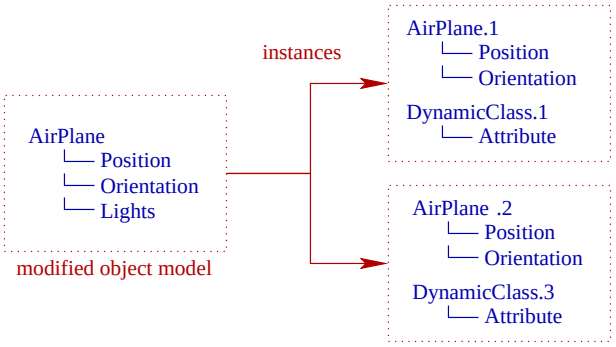

Figure 3. Dynamic Object Model

This class will be instanciated each time a new attribute is adding to an instance. The same process occurs when a new unknown class is added dynamically. For example, if the following class is defined in the Object Model :

```
(class AirPlane
   (attribute Position    best-effort receive)
   (attribute Orientation best-effort receive)
)
```

it would be impossible in the normal case to add a new `Lights` attributes while running. But if we hide how is managed the object model, the modified object model will be as seen on figure 3.

For oRis, the object model looks like the one described on the left. But an instance of this model is done (on the HLA side) by creating two class instances (an AirPlane instance and a DynamicClass instance).

4.2.2 Keeping the object model consistent

The Information Base component maintains the local database. Its objectives is to keep relationship between the oRis pair attribute/instance et the HLA pair attribute / instance.

The UML class diagram shown on figure 4 describes the relationship between the different elements. An oRis instance consist of two parts. A static part represented by the class defined in the object model and a dynamic part added each time that a new unknown attribute is added to the instance. This database is filled at initialisation. When the object model is read, static part of the agent is filled. At runtime, when new attributes are added, a new `GenericClass` is added and the corresponding attribute is associated with.

Moreover, the database have to be bidirectional. When an information is sent from oRis to the RTI, the database must retrieve the desired pair HLA attribute and HLA class instance. And when an information is sent from the RTI to oRis, the pair oRis attribute name and oRis instance name has to be retrieved. All these informations are then used by the agent's ambassadors.

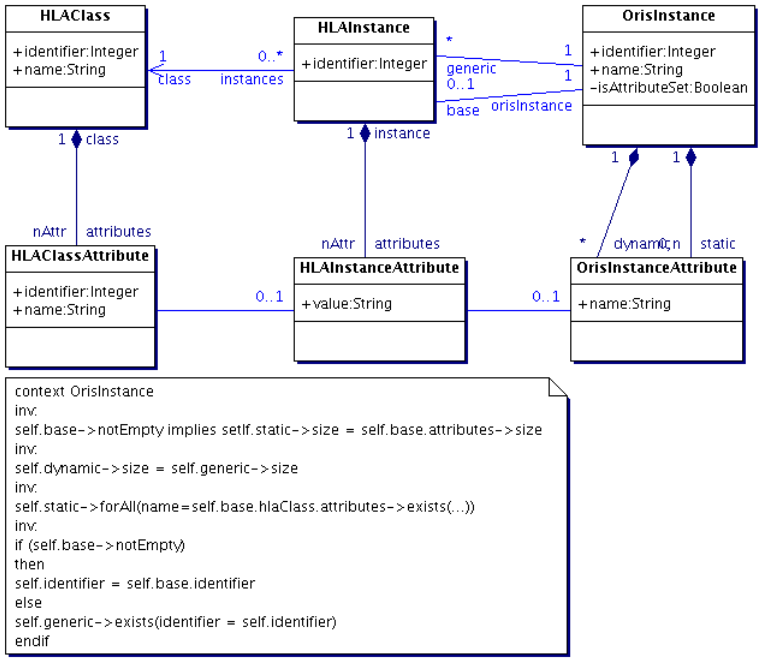

Figure 4. Information Base internals

4.2.3 Higher level ambassadors

For hiding internal management of information, the *Information Base* is surrounded by two components. One is designed for sending information (Sender) to the RTI and the other for receiving information (Receiver) for the RTI. Running of these two components is shown on the sequence diagrams[8] figure 5 and figure 6.

4.3 Extensions for the Manager

Sharing the same environment representation between the different participants is only one subset of the required features. To allow extensions to the system, writing to the RTI and reading from the RTI accessibility must stay available. Writing is made through the RTIambassador via interactions and reading is routed by the Dispatcher to the concerned specific manager (Specific Features Manager).

Indeed, many extensions are needed such as dynamic code parsing, agent migration (load balancing or session leaving) and message exchange between agents. Last point has a similar behavior as the JavaSpace usage made in [10].

5. Conclusion

In this article, we presented our architecture for building interactive and distributed prototyping applications. The HLA architecture has been integrated by providing two specific components which acts as intermediary between agents and the RTI. But, one of the trouble we encounter was the lack in instance dynamicity. By instance dynamicity, we means that the object model can't evolve during execution and is somewhat constraining for our applications. To prevent this, we created a simple database which is in charge of simulating this dynamicity.

Though being functional, one improvement would be to develop a new component for the Object Management RTI 1.3NG service[1] which take into account dynamicity.

References

[1] S. T. Bachinsky, J. R. Noseworthy, and F. J. Hodum, *Implementation of the Next Generation RTI*, Simulation Interoperability Workshop, Spring 1999.

[2] B. Blau, C. E. Hughes, J. M. Moshell, and C. Lisle, *Networked Virtual Environments*, Computer Graphics, 1992 Symposium on Interactive 3D Graphics, March 1992.

[3] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley & Sons, 2000.

[4] F. Harrouet, *oRis : in immersion through the language for virtual prototyping based on multi agents (in French)*, PhD thesis, Université de Bretagne Occidentale, December 2000.

[5] F. Harrouet, P. Reignier, and J. Tisseau, *Multiagent systems and virtual reality for interactive prototyping*, vol. 3, ISAS'99, Orlando (USA), July 31 - August 4 1999, pp. 50–57.

[6] J.-M. Naud, *Simulation Models as Components in an HLA World*, Simulation Interoperability Workshop, Spring 1999.

[7] V. Raulet, A. Nédélec, and V. Rodin, *Coupling HLA with a MAS based DVR environment*, IASTED, Applied Informatics (AI 2002), February 18–21 2002, pp. 239–242.

[8] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual (UML)*, Addison-Wesley, 1999.

[9] S. Singhal and M. Zyda, *Networked Virtual Environments - Design and Implementation*, ACM Press, SIGGRAPH Series, 1999.

[10] A. Wang, *Using Javaspaces to Implement a Mobile Multi-Agent System*, IASTED, Applied Informatics (AI 2002), February 18–21 2002, pp. 243–247.

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.

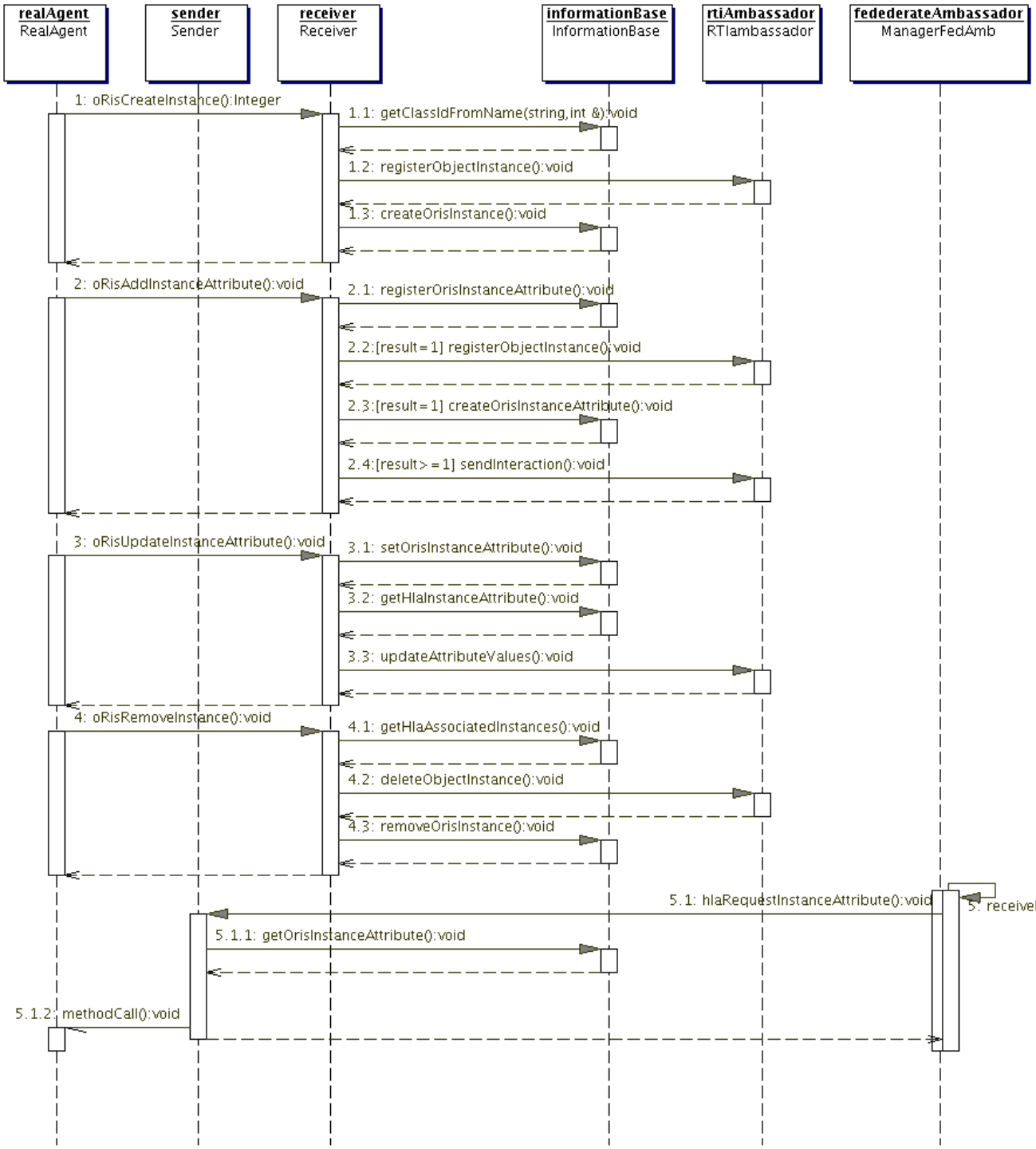

Figure 5. Sequence diagram for real interactions

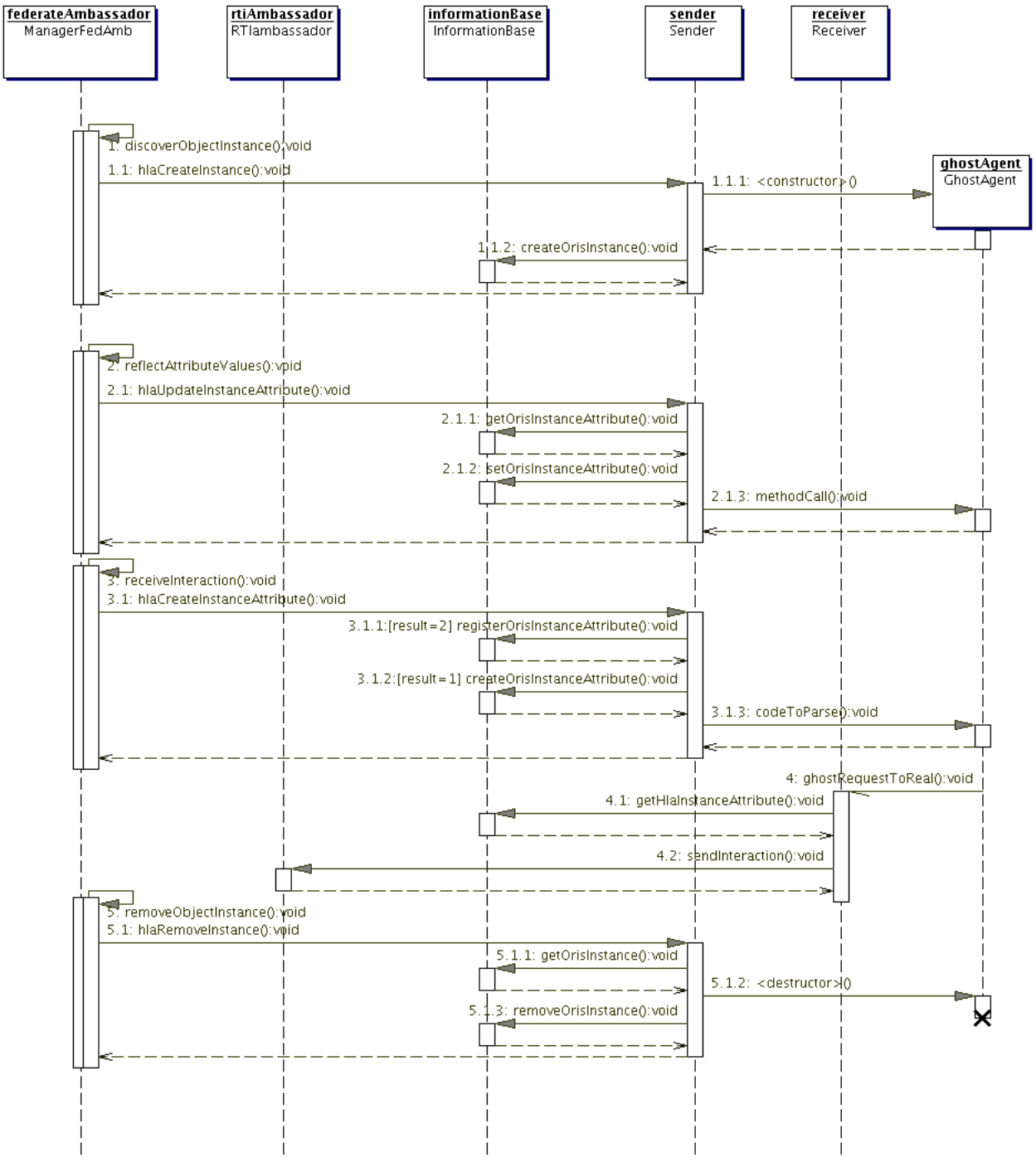

Figure 6. Sequence diagram for ghost interactions