VRML97 distributed authoring interface

Stéphane Louis Dit, Samuel Degrande, Christophe Gransart, Christophe Chaillou, Grégory Saugis

VRML97 Distributed Authoring Interface

Stéphane Louis Dit Picard* Samuel Degrande* Christophe Gransart* Christophe Chaillou*
Grégory Saugis†

*Université des Sciences et Technologies de Lille
LIFL - CNRS UMR 8022
59655 Villeneuve d'Ascq cedex, France

†France Télécom R&D - DIH/HDM/NEW
2 avenue Pierre-Marzin
22307 Lannion cedex, France

Abstract

In this paper, we present the design and implementation of the VRML97 Distributed Authoring Interface (DAI) introduced in Spin-3D, a distributed Collaborative Virtual Environment (CVE). Our proposal is a powerful interface, very close to the classical VRML97 External Authoring Interface (EAI). The DAI allows the connection of any external application with the Spin-3D CVE platform. With the Spin-3D CVE platform and the DAI, it will be easy to develop collaborative applications. We use the Common Object Request Broker Architecture (CORBA) to support distributed authoring applications. Complex collaborative applications and remote interaction introduce new considerations in the design of the DAI: we enhance the standard VRML97 EAI with new interfaces in order to easily traverse the VRML97 scene graph and limit the network overhead introduced by the remote interaction. Moreover, taking advantage of the CORBA middleware, external applications can be written with any programming language for which the OMG defined an IDL mapping.

CR Categories: I.3.7 [Computer Graphics]: 3D Graphics and Realism—Virtual Reality; C.2.4 [Computer Communication Networks]: Distributed Systems—Distributed Applications

Keywords: Virtual Reality Modeling Language (VRML), External Application Interface (EAI), Common Object Request Broker Architecture (CORBA), Collaborative Virtual Environment (CVE)

1 INTRODUCTION

In the past decades, traditional user interfaces evolved and many Virtual Environments emerged due to the rapid growth of the World Wide Web, the availability of powerful 3D graphics accelerators for PCs and high speed network devices. The Virtual Reality Modeling Language version 2.0 [Carey et al. 1997] (also called VRML97) established a standard for the description of interactive 3D scenes on the World Wide Web. This language became one of the key components of many Virtual Environments since it allows users to create 3D interactive contents without particular programming skills. Indeed, many authoring tools can produce VRML97 contents. VRML97 supports basic dynamic motion and interactivity, using *interpolator* or *sensor* nodes, and the *ROUTE* mechanism. More complex actions can be performed with programs. The first way is to use *Script* nodes to embed scripts in the VRML97 scene graph. The second way is to use the External Authoring Interface (EAI) [EAI 2002] which is similar to an Application Programming Interface (API) allowing programmers to easily develop custom applications that interact dynamically with the VRML97 scene graph. The VRML97 EAI specification represents the interface in terms of provided services and describes how to access to these services. While the specification has very general bindings, current implementations of EAI only allow Java applets to communicate with the VRML97 browser. Whereas the specification explicitly defines the possibility of remote connections, the Java applet and the VRML97 browser, embedded in the same HTML page, are running on the same machine. This paper focuses on distributed authoring.

Our proposals are widely based on the latest VRML97 EAI specification. The Distributed Authoring Interface (DAI) merges the VRML97 EAI standard and Distributed Object Technology (CORBA [OMG 2001a]). In fact, DAI runs over CORBA middleware, hiding the object distribution and the underlying implementation. Developers will not see changes between the DAI and the classical EAI, except for few programming rules specific to the CORBA programming. The DAI allows external applications to remotely control the VRML97 browser: for instance, a Personal Digital Assistant (PDA) acts as a remote controller for a VRML97 browser running on a PC. The DAI enhances the classical EAI: it allows programmers to retrieve all nodes of a VRML97 scene graph; we define new interfaces that allow an easier traversal of the VRML97 scene graph. We design these enhancements as extensions to the classical EAI thus keeping backward compatibility with it.

This paper is organized as follows: first, we review our motivations and the concepts involved in the Spin-3D CVE; secondly, we review the general concepts of the VRML97 EAI; next, we describe the overall architecture of the DAI, the EAI enhancements, the DAI implementation and the way to develop external applications using the DAI; finally, we describe a technological demonstration of the capabilities of the DAI and the Spin-3D multi-user platform.

*{louisdit, degrande, gransart, chaillou}@lifl.fr
†gregory.saugis@rd.francetelecom.com

Figure 1: A learning situation in the Spin-3D CVE.

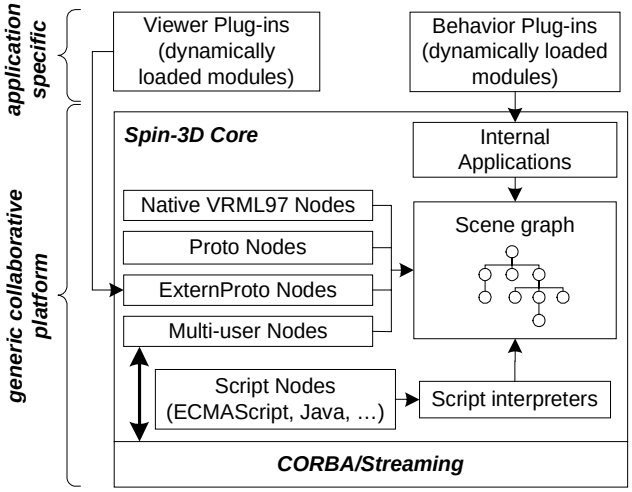

Figure 2: The Spin-3D platform architecture.

2 MOTIVATIONS

The Spin-3D project aims to propose a complete platform for Synchronous Collaborative Work making the creation of any collaborative application easier. It relies on Computer Human Interface (CHI) proposals [Dumas et al. 1999] (such as a new spatial organization taking advantage of the third dimension, a "conference table" metaphor, a 3D interaction model using two devices, one for pointing and the other for manipulating objects, and a three step interaction mechanism -select, manipulate and release-). Figure 1 shows a screen shot of the interface of Spin-3D. We use VRML97 for the description (geometry, interaction and sharing) of the 3D objects. As shown in figure 2, the Spin-3D core is built around an extended VRML97 browser developed from scratch: Spin-3D needs an extended VRML97 browser due to CHI requirements (3D pointer for designation, bounding boxes, shadows, and lightning effects missing in a standard VRML97 browser). The platform provides several mechanisms to easily develop collaborative applications:

- A "visualization plug-in" mechanism. These dynamically loaded modules extend the capability of the Spin-3D browser in allowing programmers to develop viewers for objects which can not be described with VRML97. The viewers are integrated into the Spin-3D browser using the VRML97 ExternProto mechanism. For instance, an HTML plug-in was developed in order to display HTML pages within the browser.

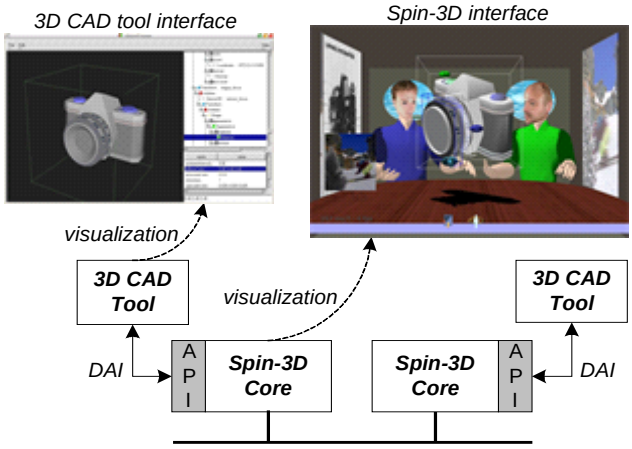

Figure 3: A multi-user 3D CAD tool with the Spin-3D collaborative platform.

- A "behavior plug-in" mechanism. Behavior plug-ins act as interface controllers and are mainly used to implement the collaborative rules which are specific to each application. For instance in a multi-user playing card game, they implement the rules of the game. They are dynamically libraries loaded by the Spin-3D browser and integrated as "internal applications". Internal applications run within the same memory space as the Spin-3D core, hence they easily access to objects contained in the scene graph.

- A VRML97 multi-user extension in order to create multi-user objects [Louis Dit Picard et al. 2002].

- A CORBA-based communication platform for the synchronization of different instances of Spin-3D and for the management of the virtual place [Louis Dit Picard et al. 2001].

These mechanisms allow programmers to write "light" collaborative applications: this is what we call "native applications". However we want to let programmers write more complex applications. This is what we call "external applications": these applications and the Spin-3D core do not run in the same memory space, moreover they can run on different computers.
For instance, if we want to develop a multi-user 3D CAD tool, we do not aim to integrate neither the 3D CAD tool into Spin-3D, nor the Spin-3D core into the 3D CAD tool. As users are familiar with the 3D CAD tool, we let them design objects within it. As shown in figure 3, each 3D CAD tool is connected to a Spin-3D core: we want to propose an API that should allow external applications to communicate with the Spin-3D core. Each change performed within the external tool is propagated to the Spin-3D core using the proposed API. The Spin-3D collaborative platform is in charge of:

- Supporting the collaboration: objects designed with the 3D CAD tool are also shown within the Spin-3D interface and so users can collaborate (discuss, manipulate, etc.) around them.

- Distributing events performed by a user within its 3D CAD tool to remote 3D CAD tools in order to ensure coherency. Remote users can in turn modify the objects using their 3D CAD tool.

The idea behind the external application notion is that third-party developers should be able to deploy existing applications in a collaborative way with only minor changes of the existing code: they only have to implement communication with the Spin-3D collaborative platform.

3 OVERVIEW OF THE VRML97 EAI

As defined by the VRML97 specifications [Carey et al. 1997][EAI 2002], there are two methods allowing programs to interact with a VRML97 scene (see figure 4):

- The first one is to use the Browser Script Interface. Internal scripts are embedded within the VRML97 scene graph using *Script* nodes. The VRML97 specification does not recommend any scripting language: it defines only the behavior of the scripting language. So scripts may be written in any programming language that the VRML97 browser supports. The *Script* node is designed to define behaviors of individual objects contained in a VRML97 file.

- The second one is to use the External Authoring Interface (EAI) mechanism. The EAI offers extended methods to access nodes and events in the VRML97 scene graph.

The VRML97 EAI is only a proposed informative appendix to the VRML97 specification. It means that browser implementations can choose whether or not to implement it. Nevertheless several standard VRML97 browsers, such as Blaxxun Contact [Blaxxun Interactive], Cortona [Parallel Graphics] or Cosmoplayer [Silicon Graphics], integrate the EAI and support Java as the primary language interface to program external applications with the EAI. Theorically any language could be used to program an external application. Nevertheless, in practice, this API is only used for communication between a VRML97 scene and a Java applet on the same HTML page and running on the same machine. Standard VRML97 browsers implement ActiveX/COM or LiveConnect interfaces to the EAI in order to allow communication between the Java applet and the VRML97 browser.

When developing a Java applet, the first operation to perform is to get a reference to the *Browser* object in order to access to the VRML97 scene. In fact, the *Browser* class is the starting point for external manipulations of the VRML97 scene: it is the only way to add new nodes, retrieve and modify existing nodes, etc. As shown in the following code, the access to the *Browser* object is performed using the *getBrowser()* method:

```
// some modules import
import vrml.eai.*;
import vrml.eai.field.*;

public class VRMLApplet extends Applet {
 Browser browser;
 public void start () {
   ...
   browser = (Browser) Browser.getBrowser(this);
 }
}
```

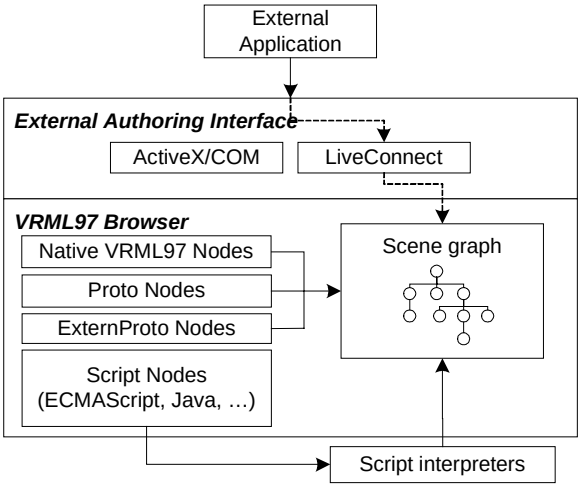

Figure 4: The architecture of a VRML97 Browser.

The *Browser* class offers a set of methods to the application programmers in order to manipulate the VRML97 scene graph:

- Methods to access nodes of the VRML97 scene graph.

- Methods to handle events to and from nodes.

- Methods to handle the VRML97 scene graph. (create new nodes, load new VRML97 scenes, etc).

The modification of a node is done by first getting its reference, using the *getNode()* method of the *Browser* class. The following code shows how to retrieve a DEF'd node:

```
Node transform;
try {
 transform = (Node) browser.getNode("TheTransform");
} catch (InvalidNodeException e) { ... }
```

The DEF name of a node contained in the VRML97 scene graph is passed as parameter to the *getNode()* method. Only DEF'd nodes contained in the scene loaded from the HTML page can be accessed from the EAI. Note that on-the-fly created nodes can also be manipulated if the external application has kept references returned at the creation step. Once we have a reference to a node, we can send events to its *EventIns*, get the current value of its *EventOuts*, or register observers on its *EventIns/EventOuts*. The following code changes the rotation value of a Transform node:

```
EventInSFRotation rotation;
float[] r = {0, 1, 0, 3.14};
try {
 rotation = (EventInSFRotation)
              transform.getEventIn("rotation");
 rotation.setValue(r);
} catch (InvalidEventInException e) { ... }
```

Observers allow programmers to monitor changes on fields: once an observer is registered to a field, it will be notified using a callback mechanism every time the field's value changes.

4 VRML97 DISTRIBUTED AUTHORING

Current EAI implementations do not offer all the possibilities of the VRML97 EAI specification: they only allow programmers to develop external applications with Java and they do not support remote connections. The Spin-3D platform has to provide a multi-language support: external applications should be written in any programming language. The proposed API also has to support remote connections. One approach is to use an ad hoc solution as in [Isakovic et al. 2002]: the X-Rooms system integrates an external API for remote simulation, authoring and interaction.
The Distributed Authoring Interface (DAI) proposes a more standardized solution: our API is very close to the classical VRML97 EAI and extends the classical EAI to networked environments by allowing external code running on a different machine to interact with a VRML97 browser. The DAI merges the standard VRML97 EAI with the CORBA Distributed Object Technology. The CORBA middleware offers us the distributed infrastructure: applications can access and interact with remote objects transparently. We choose the CORBA middleware as it is a non proprietary solution unlike the Microsoft's DCOM technology [Microsoft]. Moreover, unlike the Sun's Java RMI [Sun Microsystems], the CORBA technology is a cross-language and cross-platform distributed middleware. However the DAI could also be implementable using other distributed middlewares. Note that the DAI does not provide a byte level protocol description: network encoding relies on the CORBA specification.

4.1 Overview of the CORBA Mechanism

The Common Object Request Broker Architecture [OMG 2001a] (CORBA) is an open distributed object computing infrastructure standardized by the Object Management Group (OMG). Based on the distributed object paradigm, CORBA focuses on providing software components that may be transparently used across network by any application on any platform. CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; parameter marshalling and demarshalling; and operation dispatching. In order to perform these operations, the OMG defines several parts in the CORBA specification (see figure 5):

- The Object Request Broker (ORB) acts as a central object between the data and the application layer. Each CORBA object interacts transparently with other CORBA objects, located either locally or remotely, through the ORB. The ORB is responsible for finding a CORBA object's implementation, preparing it to receive requests, communicating requests to it and, if necessary, sending back a reply to the requester.

- The Interface Definition Language (OMG-IDL) enables the definition of methods that can be invoked on each CORBA object. A CORBA object relies on two stubs generated from its IDL interface to a given language (C, C++, Java, etc). The first one, called *client stub*, is designed for the client side, and provides the illusion that the object is local. The other one, called the *server skeleton*, is designed for the server side (i.e. the object implementation).

- A high level protocol called General Inter-ORB Protocol (GIOP) and its implementation over TCP/IP, Internet Inter-ORB Protocol (IIOP) for the definition of messages (invocation request, result message, etc), contains the Common Data Representation (CDR) to map the IDL types to their network representations and the Interoperable Object Reference (IOR) to point at an object.

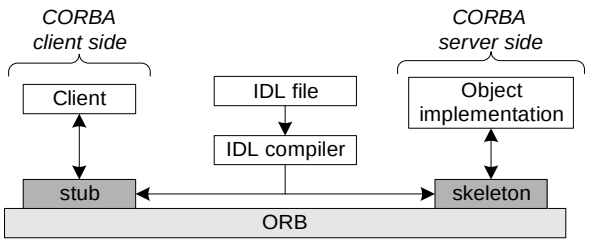

Figure 5: The CORBA specification.

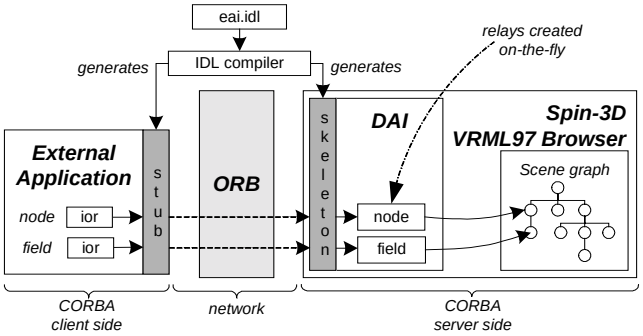

Figure 6: The DAI architecture.

The CORBA middleware is a cross-platform and a cross-language standard: it provides interoperability between objects implemented with ORB implementations from different compagnies and in different languages across network and operating systems.
Relying on the previous specifications, a distributed application works as follows: to invoke a remote method, the client makes a call to the client side stub. The client stub packs the call parameters into a request message using the IIOP wire protocol. The ORB carries the message and delivers it to the server side stub using the IOR contained in the message. The skeleton unpacks the message and calls the method on the implementation object.

4.2 Architecture

We use the model from figure 5 to design the DAI. As shown in figure 6, the Spin-3D VRML97 browser acts as the server whereas external applications act as clients. The latest EAI standard [EAI 2002] specifies the OMG-IDL interface for each service (*Browser*, *Node*, *Field*, etc). So we use it as starting point of the DAI: the compilation of the EAI OMG-IDL description generates the stubs for both sides (client and server). The server side (i.e. the Spin-3D VRML97 browser) uses the skeletons to implement methods of each interface. The client side (i.e. external applications) uses the stubs to hide the distribution of manipulated objects.

4.3 Mechanisms

The first approach is to connect all objects (nodes and fields) contained in the VRML97 scene graph to the ORB. Such an approach is not suitable because too many connected objects will reduce the ORB performance. A more suitable solution is to connect, on-the-fly, nodes and fields to the ORB, depending on requests from external applications: as shown in figure 6, the Spin-3D VRML97 browser creates objects which act as relays between the external application and the VRML97 scene graph. External applications get references (IOR) of these relays and can then invoke methods on them.

5 VRML97 EAI ENHANCEMENTS

First, let us define the operations that a complex collaborative application should have to do:

- External applications should be able to traverse the entire VRML97 scene graph and retrieve the reference of each node contained in the VRML97 scene graph.
- External applications should be able to perform operations on each node contained in the VRML97 scene graph.

For instance in order to change the color of an object, an external application should retrieve all *Material* node contained in the hierarchy and change their color attributes. Another instance, an external application should traverse the entire hierarchy in order to dump an object into an ascii file.
We enhance the services supplied by the standard VRML97 EAI in order to answer to these requirements. All enhancements are performed with the idea to keep backward compatibility with the existing EAI OMG-IDL specification. We do not modify existing interfaces of the IDL definition: we only extend existing interfaces using the OMG-IDL inheritance property.

5.1 NodePath

To access a node contained in a VRML97 scene graph with the standard EAI, the node has to be DEF'd in the VRML97 file. Once a node is DEF'd in the VRML97 file, it is available to the external application through the *getNode()* method of the *Browser* interface. But complex applications should be able to access to all nodes contained in the VRML97 scene graph, not only DEF'd nodes.
Therefore the DAI proposes a mechanism called *NodePath*, inspired by XPath [W3C]. A NodePath is a path-like which contains all intermediary nodes that lead to a given node in the VRML97 scene graph. Each node involved in the path is present in the NodePath either by its standard node type name or its DEF name:

- If the node type name is used, it appears like *"!NodeTypeName"* in the NodePath: this is the default appearance of a node in the NodePath, an exclamation mark followed by its node type name. For example, a *Tranform* node will appear as *"!Transform"* in the NodePath.
- If the node is DEF'd, the DEF name can also be used, and appears as *"DefName"* in the NodePath.

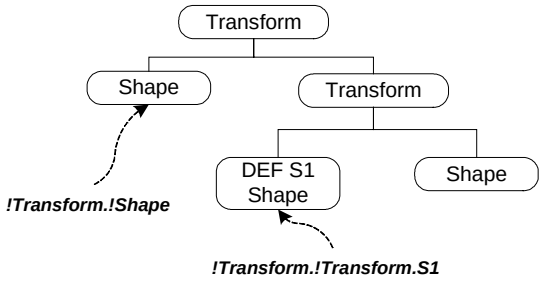

Figure 7: Examples of NodePath.

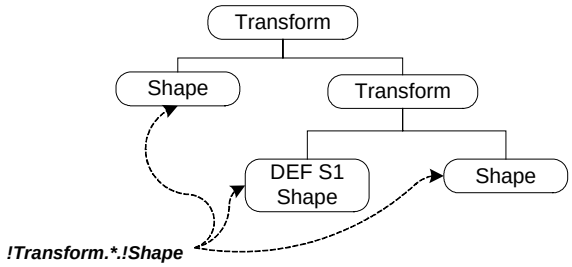

Figure 8: Wildcards used in NodePath.

The dot character (*"."*) is used as node separator. Figure 7 shows examples of NodePath. We also allow to use wildcards in the NodePath in order to replace unknown nodes in the VRML97 scene tree:

- The wildcard question mark (*"?"*) replaces one node.
- The wildcard asterisk (*"*"*) replaces any sequence of nodes.

Note that the same NodePath can designate more than one node in the VRML97 scene tree (see figure 8). As with XPath, it is possible to specify the index of node to reference. As shown in figure 9, the index of node is placed in square brackets just after the node type name (or the DEF name).

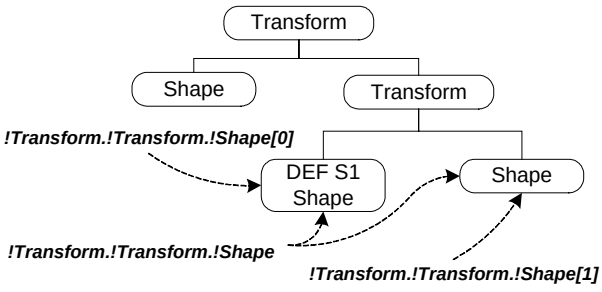

Figure 9: Indexes specified in NodePath.

5.2 The NodeSearcher Interface

As we saw before, the standard VRML97 EAI only allows to find DEF'd nodes. Such a limitation disappears when application programmers use the notion of NodePath defined

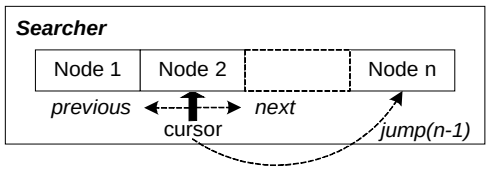

Figure 10: The *NodeSearcher* cursor used to handle navigation within the result list. On the figure, the cursor points at "Node 2" and can be moved on the left or right in order to point at another node.

```
module vrml {
 module eai {
  interface NodeSearcher {
   void   searchNode(in string npath);
   Node   getNode()
          raises (InvalidNodeException);
   Node   getNodeFromIndex(in long index)
          raises (InvalidNodeException,
                  ArrayIndexOutOfBoundsException);
   string getType()
          raises (InvalidNodeException);
   string getName()
          raises (InvalidNodeException);
   string getNodePath()
          raises (InvalidNodeException);
   long   getSize();
   void   previous()
          raises (InvalidNodeException);
   void   next()
          raises (InvalidNodeException);
   void   jump(in long index)
          raises (ArrayIndexOutOfBoundsException);
   void dispose();
  }; // End of interface NodeSearcher
 }; // End of module eai
}; // End of module vrml
```

Figure 11: IDL definition of the *NodeSearcher* interface.

before. The *NodeSearcher* interface helps application programmers to find nodes in the VRML97 scene graph. A *NodeSearcher* object will search all nodes matching a given NodePath: it is an alternative method to the standard *getNode()* method of the *Browser* interface.
As we said before, the same NodePath can designate more than one node in the VRML97 scene graph, therefore a *NodeSearcher* object manages a list of nodes, each of them matching the requested NodePath. The search is performed using a deep-first algorithm. It is not suitable to directly connect each of those nodes to the ORB: only few nodes will be used by the application and too many connected objects will reduce the ORB performance. For example, we can imagine that an application performs a search with the NodePath "*", and in such a situation the result of the search will be all nodes contained in the VRML97 scene graph. So as shown on figure 10, the *NodeSearcher* contains a cursor that points at one of the nodes contained in the list. The *NodeSearcher* interface provides a set of methods allowing external applications to move the cursor within the list. The application can then request the *NodeSearcher* object to connect nodes to the ORB.
Figure 11 shows the IDL definition of the *NodeSearcher* interface. It contains the following methods:

- The *previous()*, *next()* and *jump(i)* methods are used to move the cursor. Note that after a successful search, the cursor points at the first node contained in the list (the cursor position is equal to zero).
- The *getSize()* method is used to get the size of the result list.
- The *getNode()* method is used to retrieve the reference of the node currently pointed at by the cursor. After a call to the *getNode()* method, the node is connected to the ORB and so the application gets its reference. Note that the application is in charge of releasing interest in the node by calling the *dispose()* method of the *Node* interface.
- The *getNodeFromIndex(i)* method is used to retrieve the i^{th} node contained in the list. Note that in this case, the cursor does not move.
- The *getName()* and *getType()* methods respectively return the name and the type of the node pointed at by the cursor. They have the same behavior as the *getName()* and *getType()* methods of the *Node* interface defined in the VRML97 EAI specification. It avoids connecting the node pointed by the cursor when it is not necessary.
- The *getNodePath()* method returns the NodePath of the node pointed at by the cursor.
- The *searchNode()* method allows programmers to perform a new search: the same *NodeSearcher* object can be used several times thus avoiding to create a new *NodeSearcher* object for each search. The current list contained in the *NodeSearcher* is erased, but references of nodes obtained by the application stay valid.
- The *dispose()* method notifies the browser that the application is no longer interested in this *NodeSearcher* instance, and so the browser is free to destroy it.

In order to support the *NodeSearcher* interface defined below, we extend the *Browser* interface defined by the VRML97 EAI specification. As shown in figure 12, the *BrowserExt* interface extends the *Browser* interface and adds the support of the *NodeSearcher* interface and the notion of NodePath defined before. The *BrowserExt* interface

```
module vrml {
 module eai {
  interface Browser {
   /* Classical methods to access services provided     */
   /* by the browser: such as name, version, nodes, etc */

   Node getNode(in string name) raises (...);

   /* (...) */
  }

  interface BrowserExt : Browser {
   NodeSearcher getNodeSearcher()
                raises (ConnectionException);
  }; // End of interface BrowserExt
 }; // End of module eai
}; // End of module vrml
```

Figure 12: Extension of the *Browser* interface in order to support the *NodeSearcher* interface.

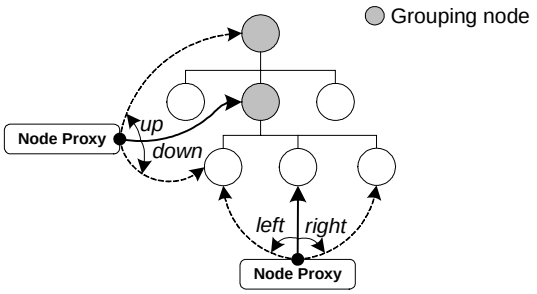

Figure 13: Node Proxy navigation.

contains the *getNodeSearcher()* method: such a method returns a *NodeSearcher* object which could be used to perform searches using a NodePath.

5.3 The Node Proxy Interface

We want to allow external applications to traverse an object VRML97 hierarchy. Services supplied by the standard VRML97 EAI do not make the traverse of the entire scene graph easier. In fact, with the standard VRML97 EAI, the only solution is to use the *getNode()* method to request each node, but it requires that all nodes of the VRML97 scene graph have a DEF name. This way will not be efficient with the DAI. In fact, as we work in a distributed environment, the external application response time depends on the ORB: the creation and connection of a *Node* object can take several milli-seconds, and thus degrading the external application interactivity.
A first approach is to use the *NodeSearcher* presented above: external applications perform a search with the NodePath "*", and the returned *NodeSearcher* will contain all nodes of the VRML97 scene graph. This solution has two drawbacks:

- All the hierarchic structure is lost.
- It requires the external application to explicitly request the *NodeSearcher* to connect each node using the *getNode()* method. As we have just explained, it degrades the external application interactivity.

A more efficient solution is to reuse an already created *Node* object, and to modify its internal state in order to point at another node of the VRML97 scene graph thus allowing navigation within scene graph. A *Node* object with such a property is what we call a *Node Proxy*. Such a mechanism is inspired by the *Traversable* [1] design pattern [Gamma et al. 1995]. A *Node Proxy* object points, at a given time, at one node of the VRML97 scene graph, and contrary to the standard EAI specification where a *Node* object statically references one node of the VRML97 scene graph, programmers can dynamically change the node pointed at by a *Node Proxy* object. As shown in figure 13, the *Node Proxy* can move *up or down* in the VRML97 scene graph, and *left or right* among children of a "grouping node".
As shown in figure 14, the *Node Proxy* interface inherits the *Node* interface in order to provide the same services as the *Node* interface (such as *getEventIn()*, *getEventOut()*, etc). It also provides methods to handle the traversal of the VRML97 scene-graph. Using the inheritance, any method

[1] Also known as the *Visitor* design pattern.

Table 1: Behavior of traversal methods of the Node Proxy interface.

Method name	Internal state (before)	Internal state (after)
parent()	node N	parent of node N
child(i)	node N	i^{th} of node N
next()	node N	right brother of node N
previous()	node N	left brother of node N

```
module vrml {
 module eai {
  interface Node {
   /* Classical methods to access node members    */
   /* (type, name, fields), and release interest. */

   eai::field::EventIn getEventIn(in string name)
          raises (...);
   eai::field::Eventout getEventOut(in string name)
          raises (...);

   /* (...) */
  }

  interface NodeProxy : Node {
   void parent()
        raises (InvalidNodeException);
   Node getParent()
        raises (InvalidNodeException);
   void child(in long index)
        raises (InvalidNodeException,
                ArrayIndexOutOfBoundsException);
   Node getChild(in long index)
        raises (InvalidNodeException,
                ArrayIndexOutOfBoundsException);
   void next()
        raises (InvalidNodeException);
   Node getNext()
        raises (InvalidNodeException);
   void previous()
        raises (InvalidNodeException);
   Node getPrevious()
        raises (InvalidNodeException);
  }; // End of interface NodeProxy
 }; // End of module eai
}; // End of module vrml
```

Figure 14: IDL definition of the *NodeProxy* interface.

which already returns a *Node* object could be used to get a *Node Proxy* object: it only depends upon the implementation. So we do not have to extend the *Browser* interface in order to support the *Node Proxy* interface: the *getNode()* method of the *Browser* interface could return a *Node Proxy* object. The same is true for the *getNode()* and *getNodeFromIndex()* methods of the *NodeSearcher* interface.
Table 1 shows effects of each method on the node referenced by the *Node Proxy*. Note that the *getParent()*, *getChild()*, *getNext()* and *getPrevious()* methods keep unchanged the node referenced by the current *Node Proxy* object (later called NP). In fact, each of these methods returns a new *Node Proxy* object which respectively references the parent of node N (we suppose that NP references node N), the i^{th} child of node N, the "right brother" of node N and the "left brother" of node N.

6 IMPLEMENTATION

The Spin-3D VRML97 browser is implemented in C++. On the server side (i.e. the Spin-3D VRML97 browser), the Distributed Authoring Interface and its enhancements are implemented in C++. The ORB that we use is Orbacus [IONA.b] from IONA. We are currently using the version 4.0.5 of Orbacus.

6.1 Establishing a connection with the Spin-3D VRML97 Browser

When the Spin-3D VRML97 browser starts, a *BrowserExt* object[2] is created and connected to the ORB. As we saw in section 3, an external application needs first to retrieve the *Browser* object. As specified by the standard EAI specification [EAI 2002], the DAI uses the *getBrowser()* method to locate the *Browser* object thus keeping compatibility with the standard VRML97 EAI. Note that using the inheritance, the *getBrowser()* method can return either a *Browser* object or a *BrowserExt* object: it only depends upon the implementation. Using CORBA facilities, different ways are available in order to implement the *getBrowser()* method:

- The first way is to use "stringified IOR" of the *Browser* object. The VRML97 browser makes the "stringified" reference of the *Browser* object available to external applications. For example, the browser IOR can be placed in a file on a http or ftp server. An application obtains the "stringified" IOR and converts it back into a *Browser* object reference, and then invokes methods on the browser.

- The second way is to use the Interoperable Naming Service (INS) [OMG 2001d]. It is a more readable format for object references that uses a URL-like syntax. The URL contains information that allow clients to rebuild the object reference (the protocol, the hostname of the server, the port on which server is listening, the unique object-key, etc.). An external application converts the URL into a *Browser* object reference, and then invokes methods on the browser. Note that, with this method, we have to identify the *Browser* object with a unique object-key. The *Browser* object-key is specified when the *Browser* object is created and connected to the ORB. So this unique object-key has to be standardized in order to allow interoperability between any external application and any VRML97 browser.

- The third way is to use the Naming Service [OMG 2001d]. The Naming Service is a standardized CORBA service that allows applications to locate objects connected on the ORB: it acts as an "object reference directory". Each object registered with the Naming Service has a unique symbolic ID, and the Naming Service maps a symbolic ID with an object reference. Note that, with this method, we have to identify the *Browser* object with a unique symbolic ID. The *Browser* symbolic ID is specified when the VRML97 browser registers the *Browser* object with the Naming Service. So this unique symbolic ID has to be standardized in order to allow interoperability between any external application and any VRML97 browser.

[2]The *BrowserExt* object can be replaced with a *Browser* object if the VRML97 browser does not support extensions such as NodePath or *NodeSearcher* objects.

Table 2: Parameters used with the *getBrowser()* method.

Method used	Parameters	Signification
using IOR	string	*Browser* IOR
using INS	string, integer	hostname, port
using Naming Service	none	none

Table 2 defines arguments that the *getBrowser()* method takes in each case described above.

6.2 Security Issues

With a standard ORB, all communications between the external applications and the Spin-3D VRML97 browser are performed by using IIOP which runs on the top of TCP/IP. Nevertheless such a solution does not offer security for communications: the IIOP protocol relies on raw TCP/IP, thereby it is not a secure protocol and does not provide authentification. Any external application can retrieve the *Browser* object and performed harmful activities to VRML97 scene graph. Therefore we have to protect the access and communication between external applications and the Spin-3D VRML97 browser. Only authorized applications should have access to the VRML97 scene graph contained in the browser. Note that, for the moment, we do not handle security at the VRML97 scene graph: authorized applications should have access to the whole VRML97 scene graph.
The idea is to replace the unsecure standard IIOP protocol by a secure protocol. The Orbacus ORB allows us to use other protocols than IIOP and proposes the Orbacus FreeSSL plug-in [IONA.a] to enable secure communications. The Orbacus FreeSSL plug-in runs on top of the Secure Socket Layer protocol (SSL) [SSL]. It provides the FreeSSL Inter-ORB Protocol (FSSLIOP) which replaces the standard IIOP. The FSSLIOP is a non-standard solution and runs only with the Orbacus ORB. The OMG is currently specifying a Secure Inter-ORB protocol [OMG 2001b].
A trustworthy authority (later called the Certification Authority) delivers an associated public and private key to each party (the external application and the Spin-3D VRML97 browser). The public key is contained into a certificate signed by the Certification Authority. A certificate has an associated private key and password. The password protects the private key and is used to decrypt the private key at runtime. Data encrypted with the private key can only be decrypted with the public key. Certificates are used to verify that they are delivered by the same Certification Authority: only certificates signed by the same Certification Authority can work together. When the external application connects to the Spin-3D VRML97 browser, the following steps are performed: first, they exchange their certificates (i.e. an exchange of public keys); next, each party (the external application and the Spin-3D browser) checks if the received certificate is trusty, i.e. it is signed by the same Certification Authority as its own certificate; finally, if the certificate check is successful, then communications between the external application and the Spin-3D VRML97 browser can start and are secured.

7 DEVELOPING AN EXTERNAL APPLICATION

Taking advantages of the CORBA specification, external applications can be written with any programming language (at least, the mapping of IDL to the chosen programming language must be defined). Currently, the OMG has defined IDL mapping for several common programming languages such as Ada, C, C++, Java, scripting language, Smalltalk, Lisp, etc. Considering the following VRML97 scene graph, we will present a sample application written in C++, Java and CorbaScript that interacts with the VRML97 browser in order to rotate the *Box* object.

```
DEF THEBOX Transform {
  children [
    Shape {
      appearance Appearance {
        material Material { diffuseColor 1 0 0 }
      }
      geometry Box {}
    }
]}
```

The following code samples show that the DAI completely hides the distributed interaction between external applications and the VRML97 browser.

7.1 Using C++

The following code is the C++ implementation of the external application which performs the animation of the box object. The syntax could be obscure to non-familiar CORBA C++ developers. Garbage collector is missing in C++ specification. The CORBA specification defines "smart pointer" variables: types having *_var* extension are smart pointers. Programmers have to take care of variable declarations.

```
vrml::eai::Browser_var browser;
vrml::eai::Node_var transform;
vrml::eai::Field::EventIn_var event;
vrml::eai::Field::SFRotation_var rotation;
float v[4], angle;

// Initialize the ORB, FreeSSL plug-in somehow

browser = vrml::eai::Browser::getBrowser();
transform = browser->getNode("THEBOX");
event = transform->getEventIn("rotation");
rotation =
    vrml::eai::Field::SFRotation::_narrow(event);
v[0] = 0.0; v[1] = 1.0; v[2] = 0.0;
angle = 0.0;
while(true) {
 v[3] = angle;
 rotation->setValue(v);
 angle = angle + 1.0;
}
```

7.2 Using Java

The following code is the Java implementation of the same external application as presented before. Note that the syntax is easier than with C++: as Java includes a garbage collector, programmers do not have to use special type to simulate garbage collector.

```
vrml.eai.Browser browser;
vrml.eai.Node transform;
vrml.eai.Field.EventIn event;
vrml.eai.Field.SFRotation rotation;
float v[4], angle;

// Initialize the ORB, FreeSSL plug-in somehow

browser = vrml.eai.Browser.getBrowser();
transform = browser.getNode("THEBOX");
event = transform.getEventIn("rotation");
rotation =
    vrml.eai.Field.SFRotationHelper.narrow(event);
v[0] = 0.0; v[1] = 1.0; v[2] = 0.0;
angle = 0.0;
while(true) {
 v[3] = angle;
 rotation.setValue(v);
 angle = angle + 1.0;
}
```

7.3 Using CorbaScript

Using the IDLScript [OMG 2001c] language specification and its implementation CorbaScript [LIFL], external applications can be written faster and easier than with Java or C++. In fact, as CorbaScript is an interpreted object-oriented scripting language, it does not require any compilation process and the syntax is less difficult than C++ or Java. All complex operations (such as ORB initialization or casting operations) are performed by the CorbaScript interpreter. CorbaScript is dedicated to CORBA environments, and scripts (later called CORBA scripts) can invoke any operation, get and set any attribute of any CORBA object. As shown in the following code sample, we can easily write small external applications in CORBA scripts form interacting with the VRML97 browser.

```
browser = vrml.eai.Browser.getBrowser()
node = browser.getNode("THEBOX")
rotation = transform.getEventIn("rotation")
angle = 0.0
while(true) {
 rotation.setValue([0.0 1.0 0.0 angle])
 angle = angle + 1.0
}
```

8 RESULTS

We developed an external application which controls a particle system. It demonstrates the capabilities of the DAI used with the Spin-3D collaborative platform. Figure 15 presents the overall architecture and the components involved in such an application. The particle system is implemented using the "visualization" and "behavior" plug-in mechanisms presented in section 2:

- On each computer, the *Particle Display* visualization plug-in is in charge of displaying particles using a vector containing positions of all particles. This vector is shared. Figure 16 shows the VRML97 file describing the particle visualization plug-in. For further informations on sharing description, the reader needs to refer to [Louis Dit Picard et al. 2002].

- On one computer, the *Particle Animator* behavior plug-in is in charge of computing the position of each particle. This plug-in updates the particle positions contained in the vector. As the vector is shared, modifications are sent to the others Spin-3D. This behavior plug-in is dynamically loaded by the Spin-3D core using informations placed into a configuration file: routes

are dynamically created between the *Particle Animator* behavior plug-in and the *Particle Display* visualization plug-in.

The external application is connected to the Spin-3D which computes the animation. We can control parameters of the particle system, such the gravity, the number of particles, etc. The controller application runs on a PDA and was written in Java. The network link between the PDA and the Spin-3D computer is achieved by a wireless network (IEEE 802.11b).

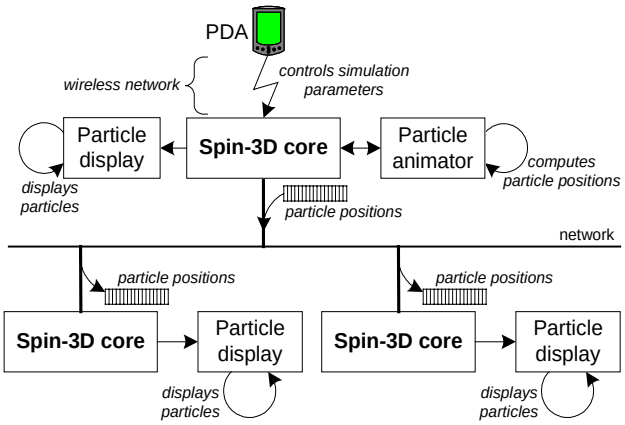

Figure 15: The "particle system" demonstration architecture.

```
#VRML V2.0 utf8

EXTERNPROTO ParticleDisplay [
  exposedField MFVec3f position
  exposedField SFColor color
  exposedField SFString texture
  exposedField SFFloat size
]["urn:file:ParticleDisplay.dll"]

SharedSet {
  mode "public"
  children [
    DEF SharedParticulePos SharedMFVec3f {
      alias "ParticleDisp.position"
      mode "public"
      how "flow"
    }
  ]
}

Transform {
  children [
    DEF ParticleDisp ParticleDisplay {
      color 0.9 0.2 0.9
      size 0.05
      texture "particle.tga"
    }
  ]
}
```

Figure 16: VRML97 file for particle visualization plug-in.

9 CONCLUSION AND FUTURE WORK

This paper presents the design and implementation of the Distributed Authoring Interface used in the Spin-3D platform. With such an API, we allow third-party developers to write easily their own collaborative applications. The DAI merges the standard VRML97 EAI and the CORBA Distributed Object Technology. Current EAI implementations do not support all the possibilities provided by the VRML97 EAI specification: theorically any language could be used to program an application using the EAI, but in practice, the current implementations only support communication between a VRML97 scene and a Java applet embedded in the same HTML page. Whereas the standard mentions the possibility of remote connections, in practice, Java applets and the VRML97 browser are running on the same machine. Taking advantage of the CORBA middleware, the DAI supports remote connections and allows developers to program external applications with any programming language. The DAI also introduces new interfaces to the classical VRML97 EAI: using the latest EAI specification and the OMG-IDL inheritance property, they are integrated as extensions. The *Node Proxy* interface handles the traversal of the VRML97 scene graph. The *NodePath* allows programmers to point at any node of the VRML97 scene graph, and the *NodeSearcher* interface is used to search and get nodes using a NodePath. This paper also presents a solution to ensure security between external application and the VRML97 browser thus protecting the browser from harmful activities. The CORBA technology allows us to replace traditional unsecure transport layer (over TCP/IP) with a secure one (using the SSL protocol). Therefore only authorized external applications could manipulate the VRML97 browser.
Currently, the DAI does not handle buffered updates to the VRML97 scene graph: the *beginUpdate()* and *endUpdate()* methods, as specified by the VRML97 standard EAI, are not yet supported. All remote method invocation are synchronous and the implementation of methods immediately modifies nodes and fields. Spin-3D is a CVE where several users can interact with each other. Spin-3D owns a lock manager which is responsible for the access to shared data. In order to write external collaborative applications, programmers should have access to the Spin-3D's lock manager. We have to extend the DAI in order to propose such a service.

Acknowledgments

The research project reported here is supported by France Télécom R&D and the regional council of Nord-Pas de Calais (France).

References

Blaxxun Interactive. See http://www.blaxxun.com.

Carey, R., Bell, G., and Marrin, C. 1997. ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97). See http://www.web3d.org/technicalinfo/-specifications/vrml97/.

Dumas, C., Degrande, S., Saugis, G., Chaillou, C., Viaud, M.-L., and Plénacoste, P. 1999. Spin: a 3D interface for cooperative work. *Virtual Reality Society Journal*.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Pattern, Elements of Reusable Object-Oriented Software*. ISBN 0-201-63361-2. Addison-Wesley.

THE EAI WORKING GROUP. 2002. ISO/IEC FDIS 14772-2:2002 Virtual Reality Modeling Language (VRML97). See http://www.web3d.org/technicalinfo/specifications/-eai/.

IONA. See http://www.ooc.nf.ca/fssl/.

IONA. See http://www.orbacus.com.

ISAKOVIC, K., DUDZIAK, T., AND KÖCHY, K. 2002. X-Rooms: A PC-based immersive visualization environment. In *Web3D 2002*.

LIFL. See http://corbaweb.lifl.fr/corbascript/.

LOUIS DIT PICARD, S., DEGRANDE, S., GRANSART, C., SAUGIS, G., AND CHAILLOU, C. 2001. A CORBA-based Platform as Communication Support for Synchronous Collaborative Virtual Environment. In *ACM Multimedia 2001, International Multimedia Middleware Workshop*.

LOUIS DIT PICARD, S., DEGRANDE, S., GRANSART, C., SAUGIS, G., AND CHAILLOU, C. 2002. VRML Data Sharing in the Spin-3D CVE. In *Web3D 2002*.

MICROSOFT CORPORATION. See http://www.microsoft.com/com/tech/dcom.asp.

OMG. 2001. The Common Object Request Broker: Architecture and Specification revision 2.5. OMG Document formal/01-09-01.

OMG. 2001. Common Secure Interoperability (CSIv2). OMG Document ptc/01-06-17.

OMG. 2001. CORBA Scripting Language Specification. OMG Document formal/01-06-05.

OMG. 2001. Naming Service Specification. OMG Document formal/2001-02-65.

PARALLEL GRAPHICS. See http://www.parallelgraphics.com/cortona.

SILICON GRAPHICS CORPORATION. See http://vrml.sgi.com.

SUN MICROSYSTEMS. See http://java.sun.com/products/jdk/rmi/.

THE TRANSPORT LAYER SECURITY WORKING GROUP. The SSL Protocol version 3.0.

WORLD WIDE WEB CONSORTIUM (W3C). XML Path Language (XPath) version 1.0. See http://www.w3.org/TR/xpath.