Application-level graphic streaming channel

Patricio Inostroza
Dpto. de Ciencias de la Computación – U. Chile
Av. Blanco Encalada 2120, Casilla 2777
Santiago – Chile
Patricio.Inostroza@dcc.uchile.cl

Jacques Lemordant
Vis / Gravir / Inria,
655 avenue de l'Europe, Montbonnot 38334, Saint
Ismier Cedex - France
Jacques.Lemordant@inrialpes.fr

Abstract. Scene Graph APIs are used to interact at the application level with a 2D or 3D scene. Examples of such APIs are the External Authoring Interface for VRML scenes or the MPEG-J scene graph API for MPEG-4 scenes. In the practice, the application is running in the scene player as an applet or MPEGlet. This paper shows how a remote scene graph API can be implemented by defining a new graphic streaming channel at the application level. We describe a simple and compact communication protocol corresponding to this streaming channel and give an example of use of this channel.

1 Introduction

The Virtual Reality Modeling Language (VRML) is a file format used to describe interactive 3D objects and worlds [3,13]. MPEG-4 has extended this language with new nodes and streaming channels for audio, video and graphics [1,13]. Concerning graphic, a binary format, called BIFS for Binary Format for Scene [1,13] has been and a protocol have been defined to stream graphics elements directly in the scene graph. MPEG-4 can handle 3D graphics objects with streaming media such as text, audio, video and images. MPEG-4 browsers, as well as authoring tools for the creation of MP4 files, are becoming to be available for different platforms, such as desktop computers mobiles or set-top boxes.

A scene graph API specifies the communication interface between a scene and an application. Examples of such API are the VRML External Authoring Interface [4] or the MPEG-J Scene Graph API [2]. Using this kind of API, an application can interact with the scene in the following ways:

- Accessing the functionality of the browser interface (i.e. to create or delete nodes)
- Sending events to eventIns of nodes inside the scene (i.e. to change positions or colors of objects)
- Reading the last value sent from eventOuts of nodes inside the scene (i.e. to get the position or color of an object)
- Getting notified when events are sent from eventOuts of nodes inside the scene (i.e. when an object is clicked)

These scenes graph APIs as specified are not limited to a local implementation. But in the practice for VRML, the browser, where the VRML scene is played, and the external application run in the same local program. In general this program is a web Browser (Internet Explorer, Netscape), the VRML browser is a plug-in and the external application is a java applet as shown on figure 1. For MPEG-4, we have the same situation, but the delivery mechanism is different. The application is called an MPEGlet[2] and streamed to the player via an MPEG-J channel. Another difference with VRML is the fact that it's less in the scope of the MPEGlet to use it for authoring.

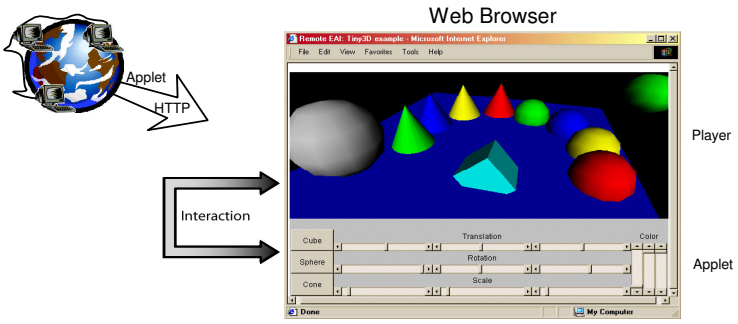

Figure 1: Using a scene graph API in the VRML case

This approach allows different levels of interaction with the scene. The user can interact directly with the world or through the applet or MPEGlet. All interactions are local ones. The interaction problem between a 2D or 3D world and a remote application is a topic which has been mainly studied in the framework of multi-user applications. Proprietary communication systems have been developed between the server and the clients [5,7,8,9]. The communication can take place internally or at the application level via the scene graph API. Figure 2 shows the interaction between a remote application (which has a graphical interface) and 3D world.

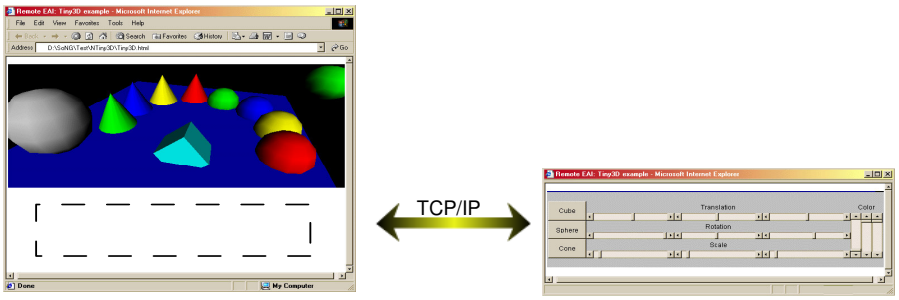

Figure 2

This paper is laid out as follow: in the section 2 we illustrate the MEG-J and EAI likeness. In section 3, we briefly introduce the design of a remote scene graph API. In section 4 we present a description of a protocol which can be used as a basis for a remote scene graph API. Our implementation is presented in the section 5. Finally in the section 6, we give an example of use of these application-level streaming channels.

2 Scene Graph API: MPEG-J / EAI

Both EAI and MPEG-J scene graph API comprise 4 elements:

1. An interface to get references on the nodes of the scene graph. This interface is named *Scene* for MPEG-J and B*rowser* for EAI.
2. An interface to manipulate the nodes: a node can be removed, stored and re-inserted in the scene graph.
3. A set of interfaces to modify the fields of a node.
4. An interface to subscribe as a listener to the events of the scene graph

List 1 and List 2 below, show an example where the same scene graph is modified by means of MPEG-J and EAI. This scene graph has a transform node that contains a sphere. In the MPEG-J example, the *Transform* node is retrieved using the S*cene* interface. *Transform* values are retrieved and new values are placed in an object that implements the SFVec3fFieldValue interface. Finally, the *Transform* node receives the new transform values.

In the same way, in the EAI example the *getNode* function of the *Browser* interface returns the *Transform* node. The EventOutSFVec3fValue interface of this node returns the current *Transform* values, and new transform values are set using the EventInSFvec3fValue interface. Finally, the *Transform* node receives the new transform values.

List 1: Scene graph examples

MPEG-4 Scene Graph	VRML Scene Graph

DEF **1** Transform {
  children [
  shape {
  geometry Sphere {}
  }
  ]
}
	DEF **myTransform** Transform {
	  children [
	  shape {
	  geometry Sphere {}
	  }
	  ]
	}

List 2: Modifying a scene graph

MPEG-J

```
import org.iso.mpeg.mpegj.scene.*;
public class MyTranslate {
  // ...
  public void translate(Scene scene)
    throws MPEGJException {

    Node node = scene.getNode(1);

    int outID = EventOut.Transform.translation;
    SFVec3fFieldValue translationEventOut =
      (SFVec3fFieldValue) node.getEventOut(outID);
    float[] translation =
      translationEventOut.getSFVec3fValue();

    // Calculate the new translation
    final float[] newTranslation = {
      translation[0] + 2,
      translation[1] + 2,
      translation[2] + 2
    };

    int inID = EventIn.Transform.translation;
    SFVec3fFieldValue translationEventIn =
      new SFVec3fFieldValue() {
        public float[] getSFVec3fValue() {
          return newTranslation;
        }
      };

    node.sendEventIn(inID, translationEventIn);
  }
}
```

EAI

```
import vrml.eai.Browser;
import vrml.eai.Node;
import vrml.eai.exception.*;
import vrml.eai.field.*;


public class MyTranslate {
  // ...
  public void translate(Browser browser)
    throws Exception {

    Node node = browser.getNode("myTransform");

    EventOutSFVec3f translationEventOut =
      (EventOutSFVec3f) node.getEventOut("translation");
    float[] translation =
      translationEventOut.getSFVec3fValue();

    // Calculate the new translation
    float[] newTranslation = {
      translation[0] + 2,
      translation[1] + 2,
      translation[2] + 2
    };

    EventInSFVec3f translationEventIn =
      (EventInSFVec3f) node.getEventIn("translation");
    translationEventIn.setValue(newTranslation);
  }
}
```

3 Architectural design of a remote scene graph API

A remote scene graph API is associated to an application-level graphic streaming channel. This streaming channel is bi-directional, with an upstream channel for scene graph events and a downstream channel to subscribe as a listener for scene graph events. At the higher level, a remote scene graph API is composed of 3 elements: a proxy, a wrapper, and a communication protocol as shown on figure 3.

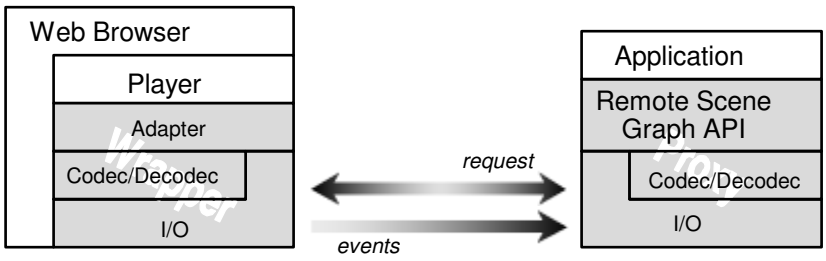

Figure 3: Architecture of remote scene graph API

The proxy or stub codifies and sends each request to the wrapper. The wrapper reads the request, decodes and executes it using the local scene graph API. If the request produces a return value, the wrapper encodes and sends it to the proxy.

4 Protocol for the graphic streaming channel

We have classified the messages as request or events messages. Requests are messages sent by the application to the world. By means of this type of message the application can, for example, create a new node or get a value of an object. Events are asynchronous messages produced in the world. The application sends a message (request) to subscribe as a listener for an event. The application will be notify if the event happens in the world. We give below the coding that we have used for the VRML External Authoring Interface, which can be seen as a superset of the MPEG-J scene graph API.

4.1 Request messages

A request message has three parts: the *object-ID*, the *method-ID*, and the *parameters* as shown in figure 4.

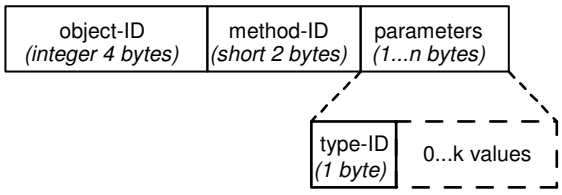

Figure 4: A request message

Object-ID is a 32-bit integer value that identifies an object, i.e. an instance of Node, EventIn or EventOut classes. The *object-ID* take the 0 value if the application calls a method of the Browser instance.

Method-ID value is a short value that identifies the method called (tables: 1.a, 1.b, and 1.c).

Parameters is a tuple (*type-ID*, *values*) where *type-ID* is a short integer and *values* is a list of simple types as described in column three of table 2. Table 2 is an extension of the set of types used by VNET, a protocol defined in a multi-users context [7]. MFVFIELD is a special type, which allows us to send a list of tuples in case a method with several parameters.

Table 1.a: Methods ID of a Node object

Method of a Node	**Method-ID** (short)
unknownMethod	0
getId	1
short getType	2
short getEventIn	3
getEventOut	4
finalize	5

Table 1.b: Methods ID of a Browser object

Method of a Browser	**Method-ID** (short)
unknownMethod	0
addRoute	1
beginUpdate	2
createVrmlFromString	3
createVrmlFromURL	4
deleteRoute	5
dispose	6
endUpdate	7
getCurrentFrameRate	8
getCurrentSpeed	9
getName	10
getNode	11
getVersion	12
getWorldURL	13
loadURL	14
replaceWorld	15
setDescription	16

Table 1.c: Methods ID of EventIn and EventOut

Method of an EventIn and an EventOut	**Method-ID** (short)
unknownMethod	0
advise	1
finalize	2
get1Value	3
getHeight	4
getId	5
getNumComponents	6
getPixels	7
getSize	8
getType	9
getValue	10
getWidth	11
newEventIn	12
newEventOut	13
set1Value	14
setValue	15

Table 2: Value part of a message

Tag	**Type**	**Encoding**	**Description**
0	SFBOOL	Byte	zero = false; no-zero = true
1	SFCOLOR	float, float, float	R, G, B colors
2	SFFLOAT	float	
3	SFIMAGE	int, int, int, [byte byte byte ..]	width, height, components, [width * height * components pixels]
4	SFINT32	int	
5	SFNODE	int	Node ID
6	SFROTATION	float float float float	3-vector of axis, angle
7	SFSTRING	utf8	utf8 string
8	SFTIME	double	seconds since January 1 1970 GMT
9	SFVEC2F	float, float	x, y
10	SFVEC3F	float, float, float	x, y, z
11	SFVOID	(none)	no data

12	SFEXCEPTION	utf8	utf8 string
13	SFEVENTIN	int	EventIn ID
14	SFEVENTOUT	int	EventOut ID
15	SFDOUBLE	double	
16	MFCOLOR	int [float float float …]	n, followed by n (R, G, B) 3-tuples
17	MFFLOAT	int [float float float …]	n, followed by n floats
18	MFINT32	int [int int int …]	n, followed by n ints
19	MFNODE	int [int int int …]	n, followed by n Node ID's
20	MFROTATION	int [float float float float…]	n, followed by n (3-vector of axis and angle) 4-tuples
21	MFSTRING	int [utf8 utf8 utf8 …]	n, followed by n utf8-encode strings
22	MFVEC2F	int [float float …]	n, followed by n (x , y) 2-tuples
23	MFVEC3F	int [float float float …]	n, followed by n (x, y, z) 3-tuples
24	MFVFIELD	int [type-1 type-2 …]	n, followed by n encoding *values*
25	MFEXCEPTION	int [uft8 utf8 …]	n, followed by n utf8 string
26	MFBYTE	int [byte byte byte …]	n, followed by n bytes

4.2 Return value

Request messages can result in a return value, which is encoded using table 2. If a method is declared to raise an exception and this exception is not raised when the method is executed, an SFVOID is returned.

4.3 Event messages

An *event* message is composed of an integer follow by a double. The integer is an id, which identifies the listener of the event, and the double is timestamp indicating the creation time of the event.

5 Connection protocol

We have defined a connection protocol to set up a graphic streaming channel between the remote application and the player. This protocol has been design to allow the set-up of multiple channels.

5.1 Protocol

The wrapper, which belongs to the player, owns a TCP/IP server socket. The wrapper waits for a remote application to open two socket connections. For each connection, the application receives a connection ID and after that, sends a message to the wrapper indicating which socket will be used for request messages and which will be used for events messages. The wrapper can accept a single connection (figure 5.a) as well as multiples connections (figure 5.b).

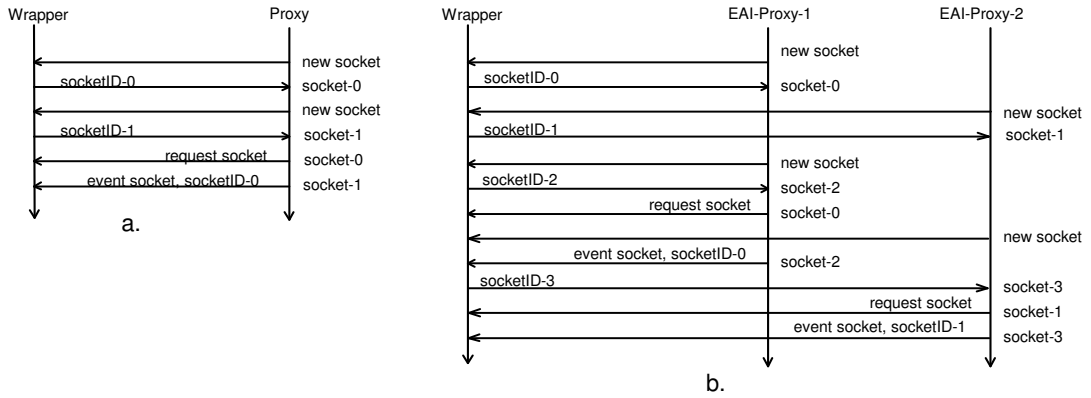

Figure 5: Sequence diagram of the connection protocol

5.2 Wrapper implementation

Our MPEG-4 3D player, called SoNG [10], is running inside Internet Explorer as an active-X component. This plug-in provides a complete java implementation of the External Authoring Interface. The wrapper is implemented as a signed java applet. The other way to implement it could have been as a COM component in C++. Figure 6 shows the different patterns of connection which are allowed.

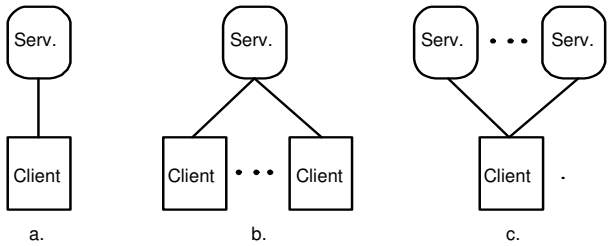

Figure 6: Connection Patterns

6 Example

We have used this application-level graphic streaming channel to build a real-time interactive 3D news service and a Javacard[11] service. The news service is using the news provider moreover.com through an XML channel and is displaying the news in a 3D scene. The design of the application is shown in figure 7. The user can choose from a menu implemented via a layer3D node, the type of news he is interested in.

When the system detects that a Javacard is plugged in the computer, a user interface composed of layer2D nodes is dynamically placed in the MPEG-4 scene. Using this interface the user can buy tickets for the theater and receive loyalty points (figure 8). Other kind of services can be plugged at run-time in the 3D world such as conferencing services (chat, whiteboard, mood aggregation opinions).

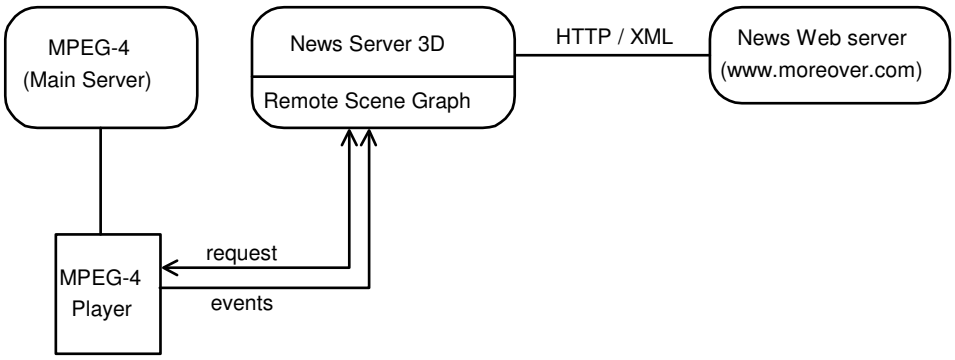

Figure 7: News service system

Figure 8: MPEG-4 Javacard UI in a theater scene

7 Conclusion

This application-level streaming channel represents an easy way to put new services at run-time in a running MPEG-4 or VRML world. The main idea behind it, is to let the player offers its scene graph API as a service to service providers. We are now using the Jini Connection technology [12,14] to let service providers find in a normative way the object implementing the remote scene graph API (the player publishes in a Jini Lookup Service its remote scene graph API). Our ultimate goal is to be able to bring an MPEG-4 player in a Jini federation both as a client and a service.

8 References

1. Overview of the MPEG-4 Standard. ISO/IEC JTC1/SC29/WG11 N4030, March 2001 http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm (Visited Sep 15, 2001)

2. MPEG-4 Systems MPEG-J. http://www.cselt.it/mpeg/faq/mp4-sys/mp4-sys-7.htm (Visited Sep 15, 2001)

3. The Virtual Reality Modeling Language. International Standard ISO/IEC 14772-1:1997 http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm (Visited Sep 15, 2001)

4. Information technology -- Computer graphics and image processing -- The Virtual Reality Modeling Language (VRML) -- Part 2: External authoring interface. http://www.vrml.org/WorkingGroups/vrml-eai/Specification/part2/ (Visited Sep 15, 2001)

5. John L. Robinson, John A. Stewart and Isabelle Labbe, "MVIP—audio enable multicast Vnet", Proceeding of the Web3D-VRML 2000 fifth symposium on virtual reality modeling language, 2000, Monterey, CA USA.

6. M. Wray, R. Hawkes, "Distributed virtual environments and VRML: an event-based architecture", Computer Networks and ISDN Systems 30 (1998) pp. 43-51

7. The VRML Interchange Protocol: VNet. http://ariadne.iz.net/~jeffs/vnet/ (Visited Sep 15, 2001)

8. Contact (blaxxun): http://www.blaxxun.com (Visited Sep 15, 2001)

9. Sony, Community Place: Vitual Society on the Web. http://www.sony.co.jp/en/Products/CommunityPlace/ (Visited Sep 15, 2001)

10. portals of Next Generation, SoNG. Information Society Technologies (IST), Project Reference: IST-1999-10192

11. Java Card Technology, http://java.sun.com/products/javacard/ (Visited Sep 15, 2001)

12. Jini Community, http://jini.org (Visited Sep 15, 2001)

13. Aoron E.Walsh, Mikaël Bourges-Sévenier. Core Web 3D. Prentice Hall, 2001.

14. Jim Waldo & the Jini Technology Team. The Jini(TM) Specifications, Second Edition Addison-Wesley, 2000.