IMPLEMENTATION OF VRML AND JAVA FOR STORY VISUALIZATION TASKS

X, Zeng, Q. H. Mehdi and N. E. Gough
Games Simulation and AI (GSAI) Centre, RIATec
University of Wolverhampton, Wolverhampton, WV1 1EQ, UK
E-Mail: x.zeng@wlv.ac.uk

KEYWORDS

Story visualisation, VRML, Java, External Authoring Interface, XML

Abstract

VRML (Virtual Reality Modeling Language) is a high level graphic language for describing 3D virtual objects and worlds on Internet. The paper considers the implementation of VRML and Java for story visualization tasks. It focuses on creating and manipulating the properties of 3D virtual objects by using natural language input. We first explore the potential of the binding between VRML and Java technologies for generating the interactive virtual worlds. We then investigate the possibilities and limitations of these methods. Finally, we present an inside look into our Story Visualiser graphic engine, discussing its internal architecture and some of the insights of Java, XML and VRML technology we have gained during its development.

INTRODUCTION

The development of interactive 3D applications is a difficult task. Unlike OpenGL and Direct3D, which are high performance graphic techniques with low-level functionality libraries that can be directly used for rendering, VRML, X3D and Java3d higher-level graphic techniques are based on scene graphs for representing a 3D interactive scene on the Web. The scene and other visual elements are described in a hierarchical structure. Apart from the rendering of a scene graph the interaction of the user with a scene graph is an important part of a 3D application (Wetering 2001). VRML is the ISO standard and has been widely accepted as a central metaphor for presentation, visualization and simulation purposes. It already has extensive usage in medicine, engineering and scientific visualization, entertainment and education. One of the strengths of VRML is it can provide interactivity in real time and there is not an excessive rendering delay. Another powerful feature of VRML is its easy extensibility and flexibility to add new node types and capabilities to the base language (Ames *et al* 1997; McCarthy and Descartes 1998) The VRML Specification defines a set of 54 built-in nodes that not only define the contents of a virtual world, but also dynamically change their properties using event sending and processing.

The work described here is part of ongoing research into a 3D story visualization authoring system (Zeng *et al.* 2002, 2003; Mehdi *et al.* 2003). Story based natural language was used as the primary input source in this system to construct a 3D virtual environment (3DVE). The natural language processing (NLP) and 3D graphic presentation techniques were integrated to allow construction and manipulation of a VRML-based scene graph in real time. In this paper, we explore the potential of the binding between VRML and Java technologies for generating the interactive virtual worlds. We also investigate the possibilities and limitations of these methods and focus on how to integrate Java, VRML and XML technologies to construct our graphic representation system.

DIFFERENT WAYS OF INTERACTION BETWEEN VRML AND JAVA

The universal acceptance of VRML as the world's first widely used non-proprietary file format for the deployment of behaviour-rich, real-time 3D applications has profound implications for how we envision information (Marrin *et al* 1997b). But VRML is very limited in terms of interactivity and it is not a general purpose programming language, and Java is not a 3D presentation language. However both were designed as web technologies and serve different goals. While Java has the ability to access VRML worlds and has dramatically changed the nature of the VRML itself while enriching and deepening the meaning of the data it encapsulates. The VRML and Java link provides a standardized, portable and platform independent way to render dynamic, interactive 3D scenes across the Internet (Brutzman 1998). For the majority of today's web projects, the marriage of Java and VRML provides the perfect solution (Marrin *et al* 1997b; Lea *et al* 1996). There are two popular approaches to using Java to extend VRML world: one is internal Java Script Authoring Interface (JSAI) and the other is External Authoring Interface (EAI).

Java Script Authoring Interface

A VRML file consists of nodes that mimic real-world objects and concepts such as various geometries and material descriptions *etc*. *Script nodes* defined within the VRML file are used to program more complex behaviour for a VRML scene. A node can signify or receive events from the user or other nodes from the scene and effect changes in the scene by using ROUTE to send events. The script is the way VRML communicates with outside world, encapsulating the Java code and providing naming conventions for interconnecting Java variables with field values in the scene. Program scripts are miniature applications that contain the logic and interfaced Java classes import the *vrml.*; vrml.field.* and vrml.node.* class* libraries to provide type conversion between Java and VRML. In this way, we can establish a link between a VRML scene and a Java application by making a handle to a Java.class file in a script node.

In this approach, the SAI provides a suite of classes and methods that enable Java to access its interface Fields and EventOuts, converting between data types and initialise the Java program and to respond to the events when the VRML browser run. In spite of its strengths, JSAI is unable to link with external data because it runs entirely within its plug-ins environment, a universe unto itself (Pesce 1997). As Kimen (1997) claimed: "VRML is- Java does- and the EAI can help".

External Authoring Interface

The EAI is a set of language-independent bindings that provides a conceptual interface between VRML scenes and an external environment. The virtual scene can be controlled via external programs or applets in the case of a web browser hosting the VRML browser. A typical web application consists of VRML browser window and additional controls in a Java applet on a same HTML page. The EAI not only allows manipulation all of the entities that internal scripts can modify. The true power of EAI is in the development of applications that incorporate VRML as only one of the elements in a presentation (Marrin et al 1997a). The creation of custom GUIs, the linking of VRML representations to external data, and even the possibility of multi-user interaction are all made possible in part because of the EAI. These are exactly the types of applications that developers have always wanted from VRML (Larsson 1999). The role and use of EAI lies in bridging and linking those Java applications to the 3D VRML scene. By using EAI to bind the data in the Java applet to the VRML world, developers are able to create a compelling cost efficient, cross-platform solution.

EAI firstly creates an object reference to the browser, using the methods of that object to locate various nodes which are named using a VRML statement DEF within a scene, once the pointer to a node is obtained. It then creates objects that encapsulate the EventIn and EventOut constructs of that node. Once that is accomplished, manipulating values within a VRML world is accomplished in the same way as SAI. Another feature of EAI is being notified when events are sent from eventOuts of nodes inside the scene. In this case the applet must subclass the EventOutObserver class, and implement the callback() method. The advice() method of EventOut is then passed the EventOutObserver. It permits handling of events from multiple sources, the source being distinguished just according to this value (Marrin 1997). Thus the applet can be notified once something has happened in the VRML world.

DISCUSSION

There is no doubt that Java offers sophisticated behaviours and enormous functionality to VRML worlds. However, whilst JSAI provides a flexible rule-based knowledge representation to handle the internal events of the virtual worlds easily, it has the disadvantage of being entirely contained within the plug-in, and references to Java also remain entirely constrained within the VRML world. This means we must be concerned with programming both Java and VRML aspects, and the performance of this combination is slow. There are few ways to bring the outside data in, so this approach is not suitable for programming control system. In contrast, the EAI has overcome these shortcomings and is useful for systemic node and dynamic outside event handling (Marrin 1998). This conceptual integration allows a Java applet to "query" a plug-in and establish a real-time event-based communication stream with it. This means that potentially any Java applet could invoke and use VRML as a visualization interface, and that VRML worlds can be manipulated through a Java-based interface (Pesce 1997).

Experience has been gained using the EAI approach the course of this research project. However, although EAI offers compelling capabilities to interact with VRML world, a number of limitations have also been encountered. EAI allows new objects to be created on the fly; the new objects can be added as child nodes to any node in the VRML world. But some methods such as CreateVrmlFromX(String, URL) are clumsy (although this also applies to the Scripting references as well) (Campbell 1998). When objects have been created through these methods, the nodes return an array of nodes and are resident in the memory. The DEF names are not accessible and getNode is unavailable on nodes that are created dynamically. We can use TouchSensor to solve the problem by asking the user to click on the object and then manipulate them, but this solution does not work well for complex objects (*e.g.* objects which are assembled from many parts). Furthermore, this unfriendly interface is not convenient in this work which uses natural language input. The second solution is to pre-code all of the possible DEFed nodes in a whole Java class and connect them to the corresponding Java variables. Also pre-define all the eventIn and eventOut for each node and attach them to Java objects with corresponding types. This method can be used for simulations that contain unchanged objects in the scene during whole operation. Clearly, this is another clumsy approach for our purposes because it introduces a massive programming overhead and makes future extension difficult.

Another limitation is that once getNode overloads DEF and any particular name might have been multiple used, then only the last occurrence of a given name is accessible (Marrin 1997). The program may thus be confused as more than one identical objects appear in the scene, so the user would fail to manipulate through being unable to get hold of the specific object.

EAI allows Java to interact with VRML world by using a Java applet. However the applet is restricted to the security sandbox, *i.e.* it is not possible to write to files on a local hard drive, and this extends the limitation of EAI. This is to protect from a malicious applet storing a virus on the computer. Even a signed applet can access local system resources as allowed by the local systems security policy. But it is not a good practice for the program extension, especially for the program that is still in the early development stage.

GRAPHIC PRESENTATION SYSTEM IMPLEMENTATION

To address the limitations of the binding between the EAI and VRML worlds we have developed a graphic presentation system. This user-extensible authoring system – termed a *Story Visualiser* - is implemented and based on VRML, Java, EAI and XML cooperation. There are three main processing layers: File Layer, DOM (Document Object Model) Layer and Control Layer. Each layer includes several modules and the architecture of the system is shown in detail in Figure 1.

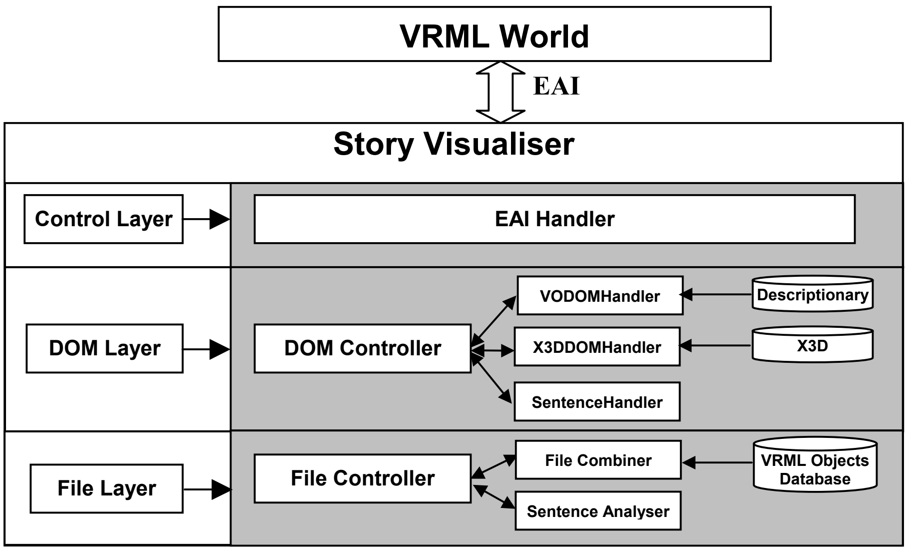


Figure 1. Story Visualiser System Architecture

The system works as follows. Beginning with the File Layer, an input sentence is analysed by the Sentence Analyser module, the XML formatted semantic representation output is extracted and a call is made to the corresponding objects from the VRML objects database. The objects are combined through a File Combiner module and converted into a set of XML files through the File Controller Layer. Then the output XML data can be operated on by the DOM Layer and passed to the EAI Handle module in Control Layer. This finally enables the Story Visualiser to construct a virtual environment. Next we describe some detail of the modular approach to discover how each of the limitations of using EAI is overcome.

The File Combiner module integrates the objects that correspond to the nouns of the semantic presentation output from the input sentence. It first reads the base VRML file *root.wrl* from the VRML database, and then read the remaining VRML objects into the memory and outputs a new combinative VRML file which contains all the objects. Once the objects have been merged into a new single VRML file, the problem of accessing the DEF names of the nodes has been addressed. However, given an increase in the number of objects and bearing in mind limitations of the natural language interface, it is necessary to find a way to restrict the DEF names of the objects. Currently, we have tackled this by generating a rule which allows sending and processing events for the specific nodes*, i.e.* change attributes of the object, such as Material node, Texture node and special related nodes (Transform node, etc). For example, the File Merge module automatically change the DEF ball as ball_1, ball_material as ball_1_material, and ball_text as ball_1_texture. If the ball appears again, the DEF name will increment by 1, *e.g.* ball_2 *etc*. Our approach is not only to formulate the DEF names of objects that provide solutions for data accessibility, consistency and efficiency, but also to define the interaction between a VRML world and a database by use of VRML and Java. Another advantage of this approach avoids the problem of possible multiple DEF names. Furthermore, because this module was written by a Java application, this means we can now read the source files, write a new file back to the local drive and address the Java applet's limitation.

XML is used for data presentation and can be used as middleware to integrate legacy systems with other applications. XML defines a standard format for representing and exchanging structured data and can be extended or embedded in Java through the use of a standard API, the Document Object Model (DOM), for managing that data, and the deployment of standard services for generating and viewing XML content. In our system, XML has been used as a major data structure for the data exchange between the different layers, *e.g.* output of the XML formatted semantic representation of the sentence to enable the system to match VRML objects and transport object depictions (*e.g.* adjectives, prepositions) to the Story Visualiser to manipulate VRML scene.

To enable the Java Applet to interact with a VRML scene through EAI, it is necessary to pre-code all the DEFed nodes and related events in the Java file in advance. This approach results in data redundancy and it is also impossible to include all the DEFed nodes in a single file. However, to extract the DEFed nodes from a VRML file is a difficult task. So in this instance, we use X3D (Extensible 3D) as the solution because it is relative easy to obtain the DEFed nodes. X3D is a 3D standard for the Web that expresses the geometry and behavior capabilities of VRML 97 using XML. It has been proposed by the Web3D Consortium since 1999 and represents the next-generation VRML. X3D is an Open Standard XML-enabled 3D file format that enables real-time communication of 3D data across all applications and network applications. However, the standard is still immature and under review by the ISO. In our system the output of the combined VRML file is converted into an X3D file using the VRML2X3D translator. We then use X3D DOM Handler to extract and store the DEFed names as a tree structure in memory to enable the EAI Handler to access and send or receive events to/from them.

COMPILATION AND EXAMPLE

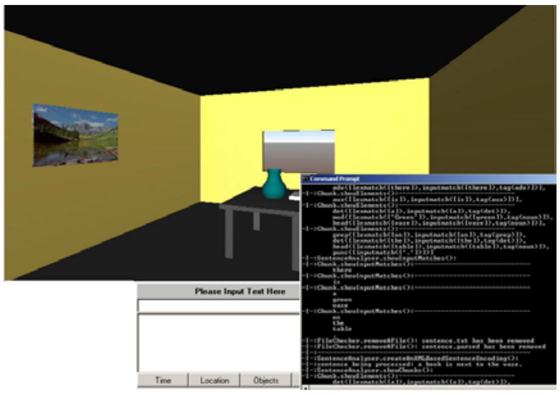
JavaSoft's JDK 1.3.1 has been used to compile the Java classes. JAXP 1.0.1 is used to parse the XML file. The EAI is using blaxxunClientSDK which is provided by Blaxxun, Inc. For the natural language part, we used NLS API which was developed by the National Library of Medicine. This implementation has been tested on a Windows 98/2000 platform using Internet Explorer 5.0 and 6.0; Netscape 4.75, 6.0.

We present here a simple example to illustrate the way of how the system works. Currently, the system enables generation of the 3D scenes and manipulation of the properties of 3D virtual objects by using natural language input. Consider a storyline that includes the following sentences to describe the environment:

There is a yellow room.
A table is in the room.
There is a green vase on the table.
A book is next to the vase.
There is a picture on the left wall.

Theses are interpreted and presented one by one. The listing shown below presents the XML formatted semantic presentation, and Figure2 shows an output 3D scene from the scene descriptions.

```
<?xml version='1.0' encoding='UTF-8'?>
<sentence-encoding>

<!-- ==== sentence 1======-->
<np reference="room_1" adjective="yellow"/>
<verb-frame  verb="is" object="room_1"/>

<!-- ==== sentence 2======-->
<np reference="table_1"/>
<verb-frame  verb="is" subject="table_1" object="room_1"
preposition="in"/>

<!-- ==== sentence 3======-->
<np reference="vase_1" adjective="green"/>
<verb-frame  verb="is" object="vase_1,table_1"
preposition="on"/>

<!-- ==== sentence 4======-->
<np reference="book_1"/>
<verb-frame  verb="is" subject="book_1" object="vase_1"
preposition="next to"/>

<!-- ==== sentence 5======-->
<np reference="picture_1"/>
<verb-frame  verb="is" object="picture_1,"
preposition="on"/>
</sentence-encoding>
```



```
        adv([lexmatch([there]),inputmatch([there]),tag(adv)])],
-[--:Chunk.showElements():-----------------------------------
        aux([lexmatch([is]),inputmatch([is]),tag(aux)])],
-[--:Chunk.showElements():-----------------------------------
        det([lexmatch([a]),inputmatch([a]),tag(det)]),
        mod([lexmatch(['Green']),inputmatch([green]),tag(noun)]),
        head([lexmatch([vase]),inputmatch([vase]),tag(noun)])],
-[--:Chunk.showElements():-----------------------------------
        prep([lexmatch([on]),inputmatch([on]),tag(prep)]),
        det([lexmatch([the]),inputmatch([the]),tag(det)]),
        head([lexmatch([table]),inputmatch([table]),tag(noun)]),
        punc([inputmatch(['.'])])]
-[--:SentenceAnalyser.showInputMatches():
-[--:Chunk.showInputMatches():-------------------------------------
        there
-[--:Chunk.showInputMatches():-------------------------------------
        is
-[--:Chunk.showInputMatches():-------------------------------------
        a
        green
        vase
-[--:Chunk.showInputMatches():-------------------------------------
        on
        the
        table
        .
-[--:FileChecker.removeAFile(): sentence.txt has been removed
-[--:FileChecker.removeAFile(): sentence.parsed has been removed
-[--:-------------------------------------------------------------
-[--:SentenceAnalyser.createAnXMLBasedSentenceEncoding():
-[--:sentence being processed: a book is next to the vase.
-[--:SentenceAnalyser.showChunks():
-[--:Chunk.showElements():-----------------------------------
        det([lexmatch([a]),inputmatch([a]),tag(det)]),
```

Figure 2. 3D scene generated from sentences

CONCLUSION

The goal of the research described here was to generate a 3DVE by using a simplified story-based natural language input. In this paper, we explained and discussed the different approach of using VRML and Java technologies to generate an interactive 3DVE. The EAI fills in the gaps between the built-in functionality of a VRML world and the programmability of Java through the use of an embedded Applet on an HTML Web page. We then presented the architecture of our system which overcomes the limitations of the EAI by integration of Java, XML, *etc* technologies. The advantage of choosing XML as the primary data structure makes the system easy to extend. Our approach also provides solutions for data accessibility, consistency and avoids redundancy. The example illustrates that the methodology works satisfactorily on generation 3D scene by manipulating the visual features of the VEs. However, the construction of the Story Visualiser system is still work in progress. It can be further improved by developing an instruction interface that will allow users to interact with the 3DVE in real time. It will also be necessary to expand the Descriptionary and VRML objects database for more complex tasks.

REFERENCES

Ames, A., Nadeau, D and Moreland, J. (1997) *The VRML Source Book*. 2nd ed. John Wiley & Sons, Inc. Canada.

Brutzman, D (1998) The Virtual Reality Modeling Language and Java. *Comm. ACM*, vol. 41 no. 6, June 1998.

Campbell, B. (1998) Extending VRML2 with Java. *Java Developer's Journal.* Vol. 3, Issue 6. SYS-CON Publications, Inc. VRMLJournal.com.

Larsson, D (1999) Linking Java with VRML97. *Technique Report*. PELAB, Department of Computer and Information science. Linköping University, Sweden.

EAIFAQ http://www.frontiernet.net/~imaging/eaifaq.html

Kimen, S. (1997) VRML Is, Java Does, And the EAI Can Help. http://vrml.sgi.com/features/java/java.html. Last access September 2001.

Lea, R., Matsude, K. and Miyashita, K. (1996) *Java for 3D and VRML Worlds.* New Riders Publishing. Indianapolis. USA.

Marrin, C., Kent, J., Immel, D. and Aktihanoglu, M. (1997a) Using VRML Prototypes to Encapsulate Functionality for an External Java Applet. http://www.marrin.com/vrml/papers/InternalExternal/Chris_Marrin_1.html. Last access November 2001.

Marrin, C. (1997) Proposal for a VRML 2.0 Information Annex. External Authoring Interface Reference. Silicon Graphics. Inc. http://www.web3d.org/WorkingGroups/vrml-hisotry/eai_draft.html. Last access August, 2002.

Marrin, C., McCloskey, B., Sandvil, K and Chin, D. (1997b) Creating Interactive Java Applications with 3D and VRML. A Silicon Graphics, Inc. White Paper. http://cosmo.sgi.com/developer.html. Last access November 2001.

Marrin, C. (1998) EAI Specification. http://www.web3d.org/WorkingGroups/vrml-eai/ExternalInterface.html. Last access November 2001.

McCarthy, M and Descartes, A. (1998) *Reality Architecture. Building 3D worlds with Java and VRML*. Prentice Hall Europe. Hertfordshire.

Mehdi, Q., Zeng, X., and Gough, N.E. (2003) Story Visualization for Interactive Virtual Environment. *ISCA 12th International Conference on Intelligent and Adaptive Systems and Software Engineering*. California.

Pesce, M (1997) VRML and Java, A Marriage Made in Heaven. http://developer.netscape.com/viewsource/pesce_vrml2/pesce_vrml2.html. Last access July 2003.

Web3D Consortium. http://www.web3d.org.

Wetering, H. (2001) Java: A Simple, Extensible, Java Package for VRML. *Proceedings Computer Graphics International 2001*, IEEE Computer Society Press, 2001

Zeng, X., Mehdi, Q and Gough, N.E. (2002) Generation of a 3D Virtual Story Environment Based on Story Description. *Proc. of 3rd SCS Int. Conf. GAME-ON 2002,* London, 2002.

Zeng, X., Mehdi, Q and Gough, N.E. (2003) Natural Language Inference Technology for Reasoning Visual Information of Virtual Environment. *Proc. of 4th SCS Int. Conf. On Intelligent Games and simulation, GAME-ON 2003,* London.