Javra : a simple, extensible Java package for VRML

Huub van de Wetering
Department of Mathematics and Computer Science
Eindhoven University of Technology
wstahw@win.tue.nl

Abstract

Javra is a Java package for handling VRML scene graphs; it operates either stand-alone or in combination with a VRML browser. The combination of Javra and a VRML browser forms an effective start for generating 3D interactive applications. With Javra a VRML scene graph can be handled: both classes for VRML nodes and methods for setting and getting the fields of these nodes are supplied. Furthermore, VRML events generated, for instance, after a user action, can be caught and handled in Javra.

The Javra node classes have an inheritance structure which allows strict compile time type checking of the construction of the scene graph. The programmers interface is intended to be simple enough to be used by students of an introductory programming course.

The node classes are generated completely automatically, resulting in a robust package. The automatic code generation can also be used to create custom Java packages for programmer-defined VRML prototypes, effectively resulting in the extension of the set of Javra nodes.

Keywords: Java, VRML, EAI, prototype, scene graph, 3D interactive graphics

1 Introduction

The development of interactive 3D applications is a difficult task. The highest performance and flexibility is achieved when libraries such as OpenGL and Direct3D are used directly for rendering, but they provide only low level functionality. A higher level is provided by, for instance, OpenInventor [8], VRML [3], and Java3D [5]. They are based on scene graphs: the scene and other aspects such as cameras and light sources are described in a hierarchical structure. Apart from the rendering of a scene graph the interaction of the user with a (displayed) scene graph is an important part of a 3D application. The problem addressed in this paper is how the development of interactive 3D applications can be simplified, such that students in an introductory course can do this job.

Our solution is to use a combination of a Java program and a VRML browser where the communication between these two is taken care of by a newly developed java package: Javra [7]. Javra is based upon the external authoring interface (EAI,[2]). It handles both the VRML scene graph as well as some of the interactions with the scene graph. The VRML browser handles the basic 3D scene interactions, like translating and rotating a scene.

Other solutions exist. The package JVerge [4] implements a similar solution as Javra. JVerge, however, is no longer supported by its creator and the Javra library is more robust in supporting the VRML nodes since these nodes are completely generated automatically. Furthermore, Javra supports generation of custom classes based on VRML prototypes, and it is based on the latest EAI standard. Java3D [5] can also handle VRML worlds but its programmers interface is not as simple as the interface of Javra. And, hence, for use in an introductory programming course it is less suitable.

Section 2 contains a system overview, in section 3 Javra nodes are introduced, in section 4 the usage of Javra is discussed. In the sections 5 and 6 examples are shown and conclusions are drawn, respectively.

2 Javra, EAI, and VRML browser

In this paper we consider applications that are built from four major components: Java, a VRML browser, EAI, and Javra. The VRML browser can be used as a Java user interface component to display VRML worlds within a Java program. The browser is interfaced to a Java program via the external authoring interface (EAI). The EAI enables programmers to manipulate a VRML world. However, EAI, is as a programming tool not very convenient since it, for instance, requires the programmer to write small pieces of VRML in the program, the syntax of which is checked during runtime. Furthermore, EAI clearly shows the VRML event model, it uses for the communication. Instead the Javra layer over the EAI gives the programmer a convenient

0-7695-1007-8/01 $10.00

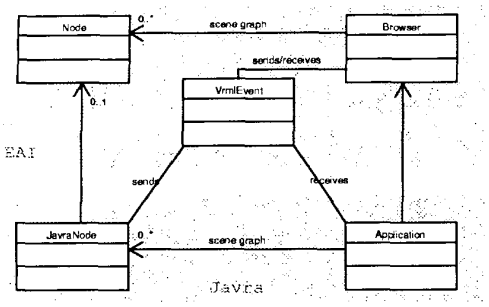

Figure 1. Simplified class diagram showing EAI and Javra relations

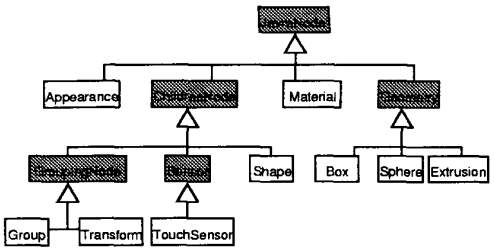

Figure 2. Javra node inheritance limited to the nodes mentioned in this paper

interface for manipulating the scene graph and handling interaction.

Figure 1 shows how Javra and EAI are coupled. Via the classes `Node` and `Browser` the EAI gives access to the scene graph in the VRML browser. In a Javra program also a scene graph (a la VRML) is created but here with so-called `JavraNodes`. This graph can be exported either to a file or a VRML browser. In the latter case the browser contains a `Node` for each `JavraNode` in the graph. Whenever a `JavraNode` is changed, by setting one of its fields a `VRMLEvent` is sent to the browser resulting in the change of the corresponding `Node` in the browser. Similarly, the browser can sent VRML events to the application when, for instance, the user clicks on an object in the scene.

3 JavraNode

The Javra package is generated automatically from a specification file (see [7]) containing an enhanced specification of each VRML node. From this file the following information is read: (a) node name, (b) field : field qualification (exposedField, field, eventOut, eventIn,...), field type (SFFloat, SFVec3f, MFNode, ...), field name, default values, and (c) class hierarchy information. For each node in the specification a Java class is created with for each (exposed) field a set and get method. The specification file is a VRML97 specification file with only small additions and alterations: A class hierarchy has been introduced and the typing of fields has been replaced by a strict typing. Part of the class hierarchy is shown in figure 2 where the dark nodes are newly introduced nodes. Note that all classes inherit from the class `JavraNode`.

Below we give the interface that is generated for the `Box` node. The box node contains only one field and both a set and a get method for it are supplied. Furthermore, `Box` inherits from `Geometry`. Consequently, it can be used in the geometry field of a `Shape` node; this fields requires a `Geometry` object due to the introduced strict typing.

```
class Box extends Geometry {
  float[] get_size();
  void set_size(float[] a);
}
```

The `GroupingNode` is one of the newly introduced nodes and it contains several fields, one of them a field containing an array of children nodes, and it may raise an event, among others, for adding new children nodes to the array. Part of its programmers interface is shown below.

```
class GroupingNode extends ChildrenNode{
 void addChildren(ChildrenNode[] a);
 void set_children(ChildrenNode[] a);
 void ChildrenNode get_children();
 ...
}
```

4 Usage

The usage of the Javra nodes is equivalent to the usage of the corresponding VRML nodes. In this section we give an overview of Javra and show some of its special features. In section 4.1 we show the building of a simple Javra scene graph, in section 4.2 we show ways to export the scene graph to either a file or a browser, in section 4.3 we handle interaction with a scene in the browser, and finally, we indicate in section 4.4 how to use VRML prototypes within Javra.

4.1 Building scene graphs

A VRML model is a set of connected nodes forming a so-called scene graph. Below the VRML code is given for a simple scene graph containing only a box.

```
Shape {
  geometry Box { size 1 2 1 }
  appearance Appearance {
   material Material { diffuseColor 1 0 0 }
  }
}
```

In the following corresponding Javra code the connections between nodes (and; hence, the scene graph) are realized by calling the corresponding set-functions.

```
Material m=new Material ();
   m.set_diffuseColor(new float[]{1,0,0});
Appearance a=new Appearance ();
   a.set_material(m);
Box b=new Box ();
   b.set_size(new float[]{1,2,1});
Shape s=new Shape ();
   s.set_geometry(b);
   s.set_appearance(a);
```

Creating a Javra scene graph requires VRML knowledge on how to construct a scene graph, specific Javra knowledge consists only of a naming convention. In general, Javra scene graph generation faithfully mimics the VRML scene graph creation where the field name `XXX` in VRML is replaced by a `set_XXX` method call in Javra.

4.2 Exporting scene graphs

Given a (Javra) scene graph we can export it to either a VRML browser that supports the EAI interface or to a file. The second option is realized by calling the print method of the top node in the scene graph. The generated VRML file will automatically make use of the DEF/USE construction of VRML to keep the resulting file small and allow the browser to optimize the rendering.

For exporting a Javra scene graph to a browser we need to obtain a handle to a browser object; this can be done by calling the appropriate methods of the EAI. Subsequently, to show the previously given box example in the browser we need the following line of code:

```
browser.replaceWorld(
   new Node[]{s.getNode(browser)}
);
```

where the `replaceWorld` method is again an EAI method call expecting an array of browser nodes (`Node[]`). To obtain such nodes the `JavraNode` s constructs a `Node` in the method call `getNode(browser)`. In this last method a lot of the error prone EAI code is handled invisibly by Javra.

Instead of replacing a world by a scene graph, nodes can also be added to an existing world (see code below). With EAI we can obtain a browser node for an object defined in the VRML world, e.g. a `Group` defined `"TOP"`. This browser node `n` can be embedded in a `JavraNode j` by the method `createJavraNode` of the class `JavraNodeFactory`. Finally, a cast to the correct Jav(r)a type may be done. Now the browser node can be manipulated via the methods of the Javra `Group` node `g`.

```
Node n=browser.getNode("TOP");
JavraNode j=JavraNodeFactory.
      createJavraNode(browser,n);
Group g=(Group)j;
```

4.3 Interaction with scene graphs

After the scene graph is exported to the browser, the dynamic aspects of the scene can still be changed. Changes are administered both in the Javra node as well as in the browser node. The changes are supplied to the browser using VRML's event model, and are handled accordingly. All this is invisible to the programmer who still uses the same methods for setting (and getting) the fields of the Javra nodes. The user can interact with the scene shown by the browser. The interactions result in `VrmlEvents` that are being tracked in the application. In Java an object implementing the `VrmlEventListener` interface of the EAI has to be supplied to the browser stating interest in receiving a certain event. `JavraNode` has a method that can be used for stating this interest in one of the events a node can generate. For example, for a `TouchSensor` and its `touchTime` event the Javra code is simply as follows:

```
TouchSensor ts=new TouchSensor();
ts.advise("touchTime",listener,data);
```

where `listener` is an object of a class that implements the `VrmlEventListener` interface and `data` is any Java object; `data` is supplied to the listener on a callback with the method `eventOutChanged` of the listener.

4.4 Extension with prototypes

Constructing scene graphs in VRML can be simplified by using prototypes; they are parameterized nodes. Prototypes typically have a relatively small amount of fields and events in their parameter list, hiding all the details of the scene graph they define. This benefit can be mimicked by a Java class that internally implements a scene graph with Javra. However, the VRML representation is more concise and clearer since it is tuned for this kind of modeling. A new

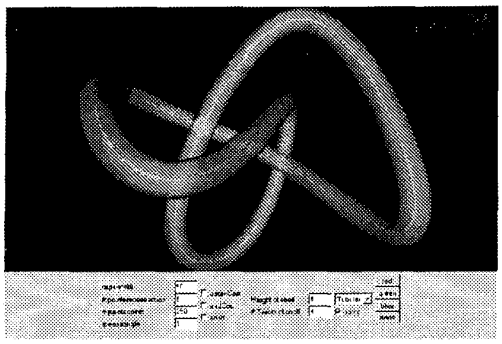

Figure 3. Playing with the `Extrusion` node

Javra class can be generated out of a VRML prototype and it can then be used in exactly the same way as the standard Javra node classes. The only prerequisite is that the browser has the prototype definition loaded. This method of using prototypes has been proposed in [6] in an EAI context but here it comes with the programmer-friendlier interface of Javra. Finally, note that generating Javra nodes directly in Java is still a good solution in the many cases where modeling directly in VRML is insufficient.

5 Examples

The figures 3 and 4 show two applets using a VRML browser [1] and the Javra package. These and more applets are available at [7]. The extrusion example (figure 3) shows an applet in which the user can input parameters for the VRML extrusion node. After selecting one of the possible spines the current world is replaced by a new world containing an extrusion object according to the current parameters and the chosen spine.

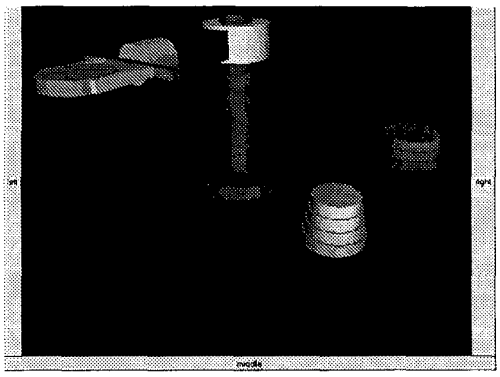

Figure 4. Robot playing towers of Hanoi

The example in figure 4 shows a robot playing towers of Hanoi; this world is generated using VRML prototypes. Both the robot and the discs in this example are constructed in VRML by means of prototypes. The robot prototype, for instance, has four parameters: the height of the arm, the angle of the arm, and the position of the left and the right finger. Its Java interface is small and all the geometry definition is done in VRML. The animation is steered by the applet by calling the methods of the generated class Robot.

6 Conclusions

The Javra package has been used extensively by students to generate VRML files in the final assignment of an introductory programming course. Javra and VRML turned out to be simple enough to be used in such a setting.

For a computer graphics specialist it may be an awkward idea to give away much of the control over the precise implementation of the rendering and to be limited to the tight structure of a VRML scene graph. But for obtaining fast results with a minimal of effort the Javra package in combination with a VRML browser seems ideal. Furthermore, the extensibility of Javra with the prototype classes introduces a flexible way of working. And finally, with the integration (in the last EAI specification) of the browser as a Java user interface component, it has become a competitor for more complex and demanding programmer interfaces.

References

[1] ParallelGraphics. http://www.parallelgraphics.com.

[2] The Virtual Reality Modelling Language - Part 2 : External Authoring Interface. ISO/IEC 14772-2, 1997. http://www.web3d.org/WorkingGroups/vrml-eai/.

[3] Rick Carey and Gavin Bell. *The annotated VRML 2.0 Reference Manual.* Addison-Wesley.

[4] Justin Couch. Jverge, a free VRML 2.0 node java class library. http://www.vlc.com.au/JVerge.

[5] M. Deering and H. Sowizral. *Java3D Specification, Version 1.0.* Sun Microsystems, August 1997.

[6] Chris Marrin, Jim Kent, Dave Inmel, and Murat Aktihanoglu. Using VRML Prototypes to Encapsulate Functionality for an External Java Applet, 1997. http://www.marrin.com/vrml/papers/InternalExternal/.

[7] Huub van de Wetering. Javra: Java & VRML assistant. http://www.win.tue.nl/~wstahw/javra, 1999.

[8] J. Wernecke. *The Inventor Mentor: programming Object-Oriented 3D Graphics with Open Inventor.* Addison Wesley, 1993.