To me, VRML has two distinct purposes: To provide a pleasing architecture to 3-D cyberspace and to provide educational and entertaining things to do within the confines of that architecture. Consider a virtual pool hall. VRML 1, as a standard, provides all the tools necessary to build an attractive and functional billiards room complete with marble floors, colorful walls, impressive artwork, cozy lighting and smoky haze.

But, visiting a virtual billiards parlor eventually loses its charm if there is no billiards to play. So, VRML 2 provides the context within which balls, racks, cues and chalk can be created and manipulated. Sensors, Timers, Interpolators, ROUTES and VRMLscript provide a primitive interface for making a pool hall respond to a virtual visitor's actions. But, as this article discusses, Java provides a more flexible yet more complicated tool for making a pool table come to life. Interpolators are great, but they must be pre-loaded with all of the possible behaviors that might take place in a world without the help of a programming language.

I enjoy creating visual 3-D objects, embedding their physics and letting them interact without any additional human intervention. Consider 16 billiard balls behaving naturally in response to a single human-initiated event: a cue stick striking a cue ball. Java makes such self-responding worlds possible. If you believe in the ability of Java to make your worlds come to life, you are halfway to making it happen. To get three-quarters of the way there, you have to choose how to add Java to your world. Then, of course, you have to learn how to program using Java.

I have given a lot of thought about how to add Java to my worlds. VRML gives us a new toy we have never had before: The ability to share visual 3-D objects through Web servers located all around the planet. Although as planetary citizens we have barely taken advantage of this opportunity, it would be nice to continue to keep the dream alive and place Java code within the confines of each VRML object. The VRML 2 standards committee continues to pursue an approach to standardizing the way Java code self-contains itself within each 3-D object in the VRML scene graph. Just as I can share VRML geometries with others on the Web using VRML with an intranode Java programming style, I can share behaviors with others as well since an object's behavior is self-contained in its Transform node. Imagine how quickly 3-D cyberspace could come to life if each VRML author spent time studying and creating a different 3-D object and made it available on the Web. With 500 or so interesting and naturally behaving 3-D objects available, all VRML authors could put together sophisticated content in much shorter periods of time.

Intranode Java certainly shows long-term promise and should deliver the kind of 3-D cyberspace many of us envision. Keep up with the latest work of the VRML Consortium to see intranode Java progress. This article, on the other hand, focuses on adding Java through an External Authoring Interface (EAI). I use the EAI to create my virtual pool halls, virtual solar systems and virtual board games. In this article I demonstrate using the EAI to create a virtual kaleidoscope. The virtual kaleidoscope project shows a good balance of using VRML 2 and Java, each for its intended purpose.

Before diving into an example, consider what makes the EAI different from intra-node Java. The EAI was designed by Silicon Graphics (SGI) for use with their CosmoPlayer VRML 2 plug-in viewer. You can read all about the EAI in the Developers' link at http://vrml.sgi.com. I think of the EAI as a presentation venue for a Java virtual machine. To me, although the EAI requires a .wrl file, the EAI is Java-centric. Intranode Java is VRML 2-centric. So, to the EAI, VRML 2 extends Java. To enable intranode scripting within VRML 2, Java extends VRML 2. This is not a subtle differentiation. These differences affect the whole thought process I follow when I develop Web-based, 3-D virtual worlds.

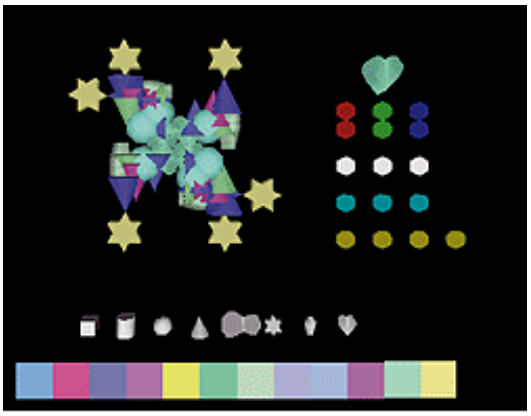

Figure 1: Kaleidoscope on first loading

Consider what is required of your audience when you use the EAI. Each participant in your virtual world must download all the Java classes you use in your world. Yet, the classes they download can include network-aware classes that help them share 3-D worlds with others. The classes they download can also include GUI controls created using the Java AWT. Using the EAI, you can provide additional controls beyond those available in their VRML viewer of choice. By focusing on Java, your world can quickly be updated to take advantage of any new Java classes or API made available by Sun or others on the Internet. But I'm sure you do not want to write your whole user interface using Java. Why not take advantage of the existing VRML viewers to give your audience the freedom to move about in your world in a manner they prefer? Using VRML 2 for

your 3-D presentation delivery is much better than any other available presentation tool on the net today. Yet, a 3-D API from Sun shows much promise for the future. Check out http://java.sun.com/products/java-media/3D/forDevelopers/3Dguide/j3dTOC.doc.html to see Sun's 3-D API specification.

One final note before I dig into the details of using the EAI. There is no reason you can't use both the EAI and intranode scripting in the same 3-D virtual world. SGI builds the EAI with flexibility in mind.

Using The EAI

The process of using the EAI is best understood through example. Simplifying a bit, the overall process goes as follows:

1. Create the VRML 2 world of your dreams following the syntax of VRML 2.
2. Add additional empty Transform nodes wherever you like to add new objects dynamically in response to your audience's actions.
3. Uniquely name the nodes you want to change dynamically with Java using the DEF keyword.
4. Extend Java's Applet class to create your own class that will be aware of your VRML 2 file.
5. Create an HTML document to connect your VRML 2 world from step 3 with your class in step 4.
6. Follow the architecture of the EAI to connect your class from step 4 to the browser object of the Web browser.
7. Follow the architecture of the EAI to provide attributes in your class in step 4 that map to the DEF nodes of step 3.
8. Add an additional attribute for each change you want to make to each attribute in step 7 (translation, rotation, scale, color, etc.).
9. Add an additional attribute for each attribute in step 7 you want to be mouse-click aware.
10. Create an event handling routine for each attribute you specify in step 9.
11. Pre-package as text strings any new VRML nodes you may want to place into the world in response to user actions.
12. Create methods (or additional classes with methods) which use the attributes of your class from step 4 to dynamically change each attribute based on the state of the virtual machine.

Then, the Java Virtual Machine can run on its own as you rely on the EAI to pass events to your Applet-extended Java class. You just have to provide the interesting event handling routines and embed the realistic physics in each object you want to behave on its own. Consider putting each of those objects into separate classes should you want to reuse them in other projects. As your project gets more complex, create an Animator class to manage the interactions between objects and a Timer class to provide control over how fast things run. Long term, you can create a Java server and connect your world to others using Java networking classes. Java makes networking worlds relatively easy compared to other programming languages.

Using The EAI in a Virtual Kaleidoscope

Driving the EAI to its breaking point requires a tremendous programming effort. First things first: Here I provide you with an intermediate use of the EAI so you can see much of its design in use. My Kaleidoscope world works well on 486 or Pentium-based PCs and provides the user with an interesting array of tools to use to change the world. The Kaleidoscope world handles some behavior in the VMRL 2 file, yet extends user interactivity through Java. Basically, it is a good starting point for your study of the EAI.

The kaleidoscope, when first loaded in the CosmoPlayer VRML 2 viewer (downloadable from http://vrml.sgi.com), appears as shown in Figure 1. The kaleidoscope, in the upper left of Figure 1 consists of eight shapes chosen randomly from a shape palette. You design the eight shapes you want to use in the kaleidoscope and create them as standard VRML 2 shape nodes. Each shape appears in a random color. The same eight shapes appear as a horizontal palette below the kaleidoscope. Below the eight shapes palette is a color palette. To the right appears a control panel with 16 buttons. The buttons provide tools a user can use to interact with the kaleidoscope. Figure 2 shows the significance of each control panel button.

A user interacts with the kaleidoscope by clicking on a shape in the kaleidoscope, redesigning the shape using the shape palette, color palette and control panel, and then putting the new shape into the kaleidoscope in place of the old. The three white buttons from left to right allow a user to replace a shape, start the kaleidoscope animation and stop the kaleidoscope animation. The rest of the control panel, from top to bottom allows users to modify a shape they are designing by decreasing the intensity of color, increasing the intensity of color, increasing scale or rotating the shape. Figure 2 recaps the significance of the control panel buttons. Note that the left-most white button will choose a random new shape if a user clicks on it without designing his or her own new shape first.
You can build the kaleidoscope by following the steps outlined above. I will walk you through the basic steps here, but you should study the complete project code which includes the VRML 2 file (**K.wrl**), Java file (**K.java**) and HTML file (**K.html**). Not all lines of code are covered by this condensed explanation.

1. Create the VRML 2 world of your dreams following the syntax of VRML 2.
Every visible object you see in Figure 1 exists as a DEF-defined Transform node in a single VRML 2 file. Note that each Shape node is a simple primitive VRML 2 shape except for the shapes you use in the kaleidoscope. Those eight shapes can be any shapes you want to design. I just happened to include a star, heart, guitar body and vase as examples. Each Transform node in the Kaleidoscope world contains a TouchSensor node in order to connect the shape to an event handling routine that will affect the world upon a user's mouse click.

In my VRML 2 file, K.wrl, I use the following Transform naming conventions: The color controls have names that begin with TC, the white animation control buttons begin with T0, the scale controls start with TSCALE, the rotation controls start with TROT, the shape palette controls have a T followed by a shape name (such as TGUITAR) and the color palette items start with a CM.

Each kaleidoscope control is a Transform node similar to the following, which shows a color palette control's Transform node. Note that you can use the USE keyword to reuse geometry or appearance field nodes as I do to reuse the geometry for each color palette square:

```
DEF CM2 Transform { children [
     DEF TSC2 TouchSensor {}
     Shape {}
]  translation 2 0 0 },
```

The eight shapes in the kaleidoscope move according to simple OrientationInterpolator nodes and two of the eight nodes move according to PositionInterpolator nodes as well. You should put the movement in the VRML 2 file when it repeats over and over. You should put the kaleidoscope's behavior in the Java file instead if it lacks a highly repetitive motion. The kaleidoscope contains two separate grouping Transform nodes (named T1 and T2) that contain eight nodes each. Transform nodes T3, T4, T5 and T6 reuse T1 and T2 to simulate a mirror reflecting the images in the kaleidoscope. ROUTE statements connect the animation start and stop control buttons to the TimeSensor (named Time_T) and interpolators needed for the animation.

2. Add additional empty Transform nodes wherever you might like to add new objects dynamically in response to your audience's actions.

The Transform nodes that make up the kaleidoscope (their names start with a W) do not contain any shapes when the VRML 2 file initially loads in the Web browser, yet they do contain TouchSensor nodes. Transform node W9 will contain the shape of the node the user is designing at any time. W9 also contains no shape when the world initially loads.

3. Uniquely name the nodes you want to change dynamically with Java using the DEF keyword.

Note that all the nodes I want to change have been given unique names in step 1 above.

4. Extend Java's Applet class to create your own class which will be aware of your VRML 2 file.
After importing all the classes from the Java language and EAI vrml packages you need in order to compile your code, you extend the Applet class with your own class. For Kaleidoscope world, I created a class I name K with the line:

```
public class K extends Applet implements EventOutObserver {
```

5. Create an HTML document to connect your VRML 2 world from step 3 with your class in step 4.
The HTML file, K.html (see Listing 1), connects the K.wrl file with the Java code using the HTML tags:

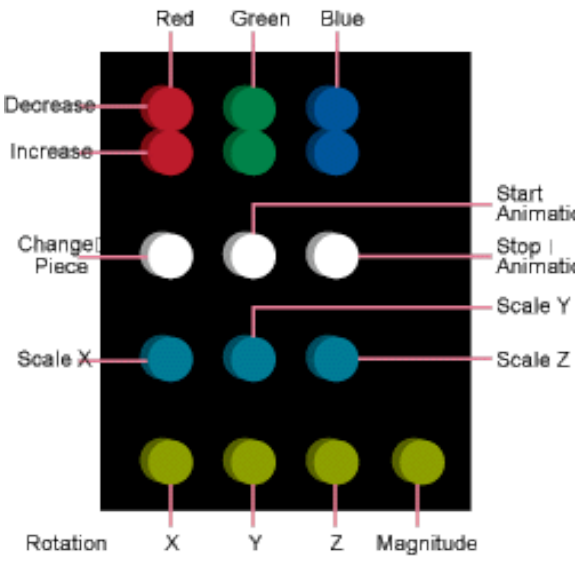

Figure 2: Control panel buttons

```
<center><embed src="k.wrl" border=0
height="500" width="500"></center>
<applet code="K.class" mayscript></applet>
```

6. Follow the architecture of the EAI to connect your class from step 4 to the browser object of the Web browser.

I connect to Netscape Navigator with these four lines:

```
JSObject win = JSObject.getWindow(this);
JSObject doc = (JSObject)
win.getMember("document");
JSObject embeds = (JSObject)
doc.getMember("embeds");
browser = (Browser) embeds.getSlot(0);
```

7. Follow the architecture of the EAI to provide attributes in your class in step 4 which map to the DEF nodes of step 3.

For Kaleidoscope world, you need 14 Node type attributes, some of which can take advantage of array indexing. The variable named root[] provides an

indexed Node for each shape in the kaleidoscope. The variable named shape[] provides an indexed Node for the shape of each root[] node. The next six Node arrays create attributes for the kaleidoscope nodes, color palette nodes, shape palette nodes, scale control nodes and rotation control nodes. For example, the rotation controls are connected to the K class through the line:

```
Node snsrro[] = {null,null,null,null};
```

The last six Node attributes set up attributes for the color controls.
Each attribute connects to the VRML 2 file through the getNode() method of the browser object. For example, in the init() method of the K class, you can connect the increase green color control to its appropriate Transform node using the line:

```
snsrg = browser.getNode("TSCG");
```

8. Add an additional attribute for changes you want to make to each attributes in step 7 (translation, rotation, scale, color, etc.).

For Kaleidoscope world I decided to package each change of translation, rotation, scale or color in a string which follows the syntax of valid VRML 2 nodes. So, instead of changing each design feature separately, I name removeChildren[] and addChildren[] attributes to allow me to remove VRML nodes and replace them with new ones in a single line of code. Both are of type EventInMFNode. The EAI provides types for changing translation, rotation, scale or color one at a time as well without having to create VRML 2 syntax strings.

9. Add an additional attribute for each attribute in step 7 you want to be mouse-click aware.

You must also create a separate EventOutSFTime type attribute to enable each touch sensor you want users to be able to use in your world. These attributes connect to touch enabled timers in the init() method of the K class as with the line tTimeg = (EventOutSFTime) snsrg.getEventOut("touchTime"); which connects the touch sensor from the green increase intensity control to its appropriate timer.

10. Create an event handling routine for each attribute you specify in step 9.

You connect your time attributes to your event handling routines through the use of an integer as with the line tTimeg.advise(this, new Integer(25));

Then, in the callback() method of your K class (all EAI applications must override this exact callback() method in order to work correctly), you create a routine that will run each time the appropriate touch sensor is activated. In the case of the green intensity control, the event handling routine appears as in the following. Assuming you have assigned the active touch sensor the value 25, the objgreen variable increases by .1, which makes the node being designed greener (if, of course, the node is not already at maximum green intensity):

```
else if (whichNum.intValue()==25) {
        objgreen = objgreen + .1;
        if(objgreen>1) {objgreen = 1;}
}
```

1. Pre-package as text strings any new VRML nodes you may want to place into the world in response to user actions.

For Kaleidoscope world, you create eight different shapes you want to add to the kaleidoscope in different combinations. Each of these eight shapes is associated with a shape attribute which converts from a simple string using the createVrmlFromString method of the browser class. You remove unwanted nodes, for example shape[8], using lines of code like removeChildren[8].setValue(shape[8]); and add new nodes using lines of code like addChildren[8].setValue(shape[8]);

You must make sure you remove the exact same node as exists in the VMRL 2 scene graph at any time. Otherwise, you cannot add a new node in its place.

2. Create methods (or additional classes with methods) which use the attributes of your class from step 4 to dynamically change each attribute based on the state of the virtual machine.

In Kaleidoscope world, much of the processing works on manipulating strings which create VRML 2 Transform and Shape nodes. All translations take place within the VRML 2 file which makes sense since the kaleidoscope's movement is very repetitive in nature. Much of the power of the EAI takes advantage of moving VRML 2 objects according to complex logic embedded in Java classes. I would use Java to create kaleidoscope behaviors that were based on real world physics. In that case, my virtual machine would have to perform collision detection and move the shapes according to the logic that responds to collisions.

Note: Kaleidoscope world was created specifically for the EAI included with Silicon Graphics' CosmoPlayer 1.0 release of their VRML viewer. I compiled my Java code using JDK 1.0.9. In the 1.0 implementation, I perform multiple EAI changes to a node in response to a single touch event. In version 2.0 of CosmoPlayer, multiple changes to a single node within a single event handling routine are not guaranteed to be handled by the viewer in the order they appear in the Java code. So, a removeChildren.setValue() method is not guaranteed to take place before the related addChildren.setValue(). To avoid any console warnings, you could add another white control button whose event handling routine would contain the necessary remove node functionality. Then, the add node routine could exist separate from the remove node routine.

Conclusion

The EAI fills in the gaps between the built-in functionality of a VRML 2 viewer, model specification of VRML 2 file syntax and programmability of Java through the use of an embedded Applet on an HTML Web page. Once a VRML 2 file has been read into a Java class structure, the world model can interact with a Java Virtual Machine capable of all kinds of new, creative processing including processing on multiple machines across a network. Creating interactive worlds with the EAI is thus open-ended, yet every Java class you involve in the processing must be delivered to each user. If your user's VRML viewer already contains the processing logic to perform an action to your virtual world, you should attempt to use the appropriate VRML 2 mechanism for enabling that action. Java through the EAI is appropriate for extending a VRML viewers capabilities perhaps only until the next VRML viewer version that contains the necessary enhancement.

Complete project code for Kaleidoscope may be found at www.sys-con.com/vrml.

About the Author

Bruce Campbell is a virtual reality and human interface research scientist working at the Human Interface Technology Laboratory at the University of Washington in Seattle. He enjoys teaching groupware and Web-related technologies through writing books and lecturing in front of a live audience. Bruce can be reached at bdc@hitl.washington.edu