Contrasting SGI's Behaviors proposal and others

This document describes the issues we have with other behaviors proposals (mainly the proposal from Mitra, Sony, SDSC and others: http://earth.path.net/mitra/papers/vrml-behaviors.html; I'll refer to that as the "MSS proposal" in this document) and describes why we think our proposal (http://reality.sgi.com/employees/gavin_engr/vrml/Behaviors.html --which I'll refer to as the "SGI proposal") is stronger.

Gavin Bell, gavin@sgi.com, Nov 1 1995

Performance

Giving the person looking at VRML worlds a good experience is our number 1 priority. And we believe that the number 1 problem with VRML today is performance-- VRML must run well on everything from a PC to a RealityEngine2 to truly be successful.

We believe that the more information given to the browser, the better the browser will be able to optimize the scene. Some specific features that will lead to bad performance:

Arbitrary effects on scene

The MSS proposal allows a script to resolve any named node during the execution of the script. Because the browser does not know which fields of which objects a script might change, it will not be able to do any of the following things:

1. Convert the scene graph into a different internal representation better suited for whichever rendering library is being used.
2. Decide not to run the script because its results are irrelevant (for example, a script driving an animation that cannot be seen does not need to be run).
3. Decide which parts of the scene can be optimized based on which parts it knows will not change.

It is naive to expect that only a few nodes in the scene will be given names; many authoring packages give every single object a name (either user-defined or system-generated) to allow the author to refer to objects.

The MSS proposal adds the CULL_IN/CULL_OUT and LOD_IN/LOD_OUT events to try to address the irrelevant script problem. Script creators will not do the right thing with these events when it is much easier to do nothing. Scripts will work well until worlds start getting larger than the tiny, toy-sized worlds common in the VR community today. The SGI proposal gives browsers enough information so that they can take care of the irrelevant script problem automatically.

Excessive script invocation

Exactly when scripts are executed isn't specified in the MSS proposal, but assuming scripts are executed whenever an 'event' happens, their proposal will execute script methods more often than necessary. For example, a script that is detecting and responding to the modification of several fields (using FieldEvents) will, I assume, be executed whenever each of the fields change (since each of the FieldEvents has a separate actionMethod). If two of the fields change (perhaps because of the execution of some other script), then the script will be executed twice. In the SGI model, a script that depends on two inputs will (by default) be executed only once, even if multiple inputs have been changed since the last time the script was executed.

The same is true for single inputs that change multiple times in between execution of the script; by default, the script is run only once. In the SGI proposal, the author must tell the browser that they care about every single change to an input or set of inputs to get a script executed multiple times. In most cases a script does not care about the history of changes to things it depends on, it only cares about the current state of the things it depends on. For example, a script that keeps an object tracking the mouse (a DragSensor in the SGI proposal, a Drag callback in the MSS proposal) will typically not care about intermediate mouse positions, but instead will only care about the latest position.

Executing scripts multiple times is not a really big problem until you get chains of scripts that depend on the values set by other scripts (which depend on values set by yet other scripts...). In those more complicated cases, executing scripts multiple times becomes a HUGE performance problem, since authors can easily create networks that have O(2^N) behaviors, where N is the number of chained scripts (it is actually M^N, where N is the number of chained scripts and M is the average number of values that each depends on).

For example, imagine scripts A, B, and C. Script A (run because of some user event) sets two values in the scene graph (perhaps the position and rotation of a transformation T1). Script B has FieldEvent callbacks so it is notified whenever either the position or rotation of that transformation changes, and sets the position and rotation of some other transformation T2... which Script C is interested in. If we play computer and trace the flow of events:

```
A sets T1.position
  Event "T1 position changed" queued for B to process
A sets T1.rotation
  Event "T1 rotation changed" queued for B to process
B's "T1 position changed" method is called
B sets T2.position and T2.rotation
  Events "T2 position" and "T2 rotation" changed queued for C
B's "T1 rotation changed" method is called
B sets T2.position and T2.rotation (again)
  Events "T2 position" and "T2 rotation" queued for C (again)
C is executed 4 times, for the 4 events waiting for it
```

A is executed once, B is executed twice, and C is executed 4 times. The MSS proposal could be modified to avoid this, perhaps by looking in the queue of waiting events and replacing old events from the same EventHandler with new events. That would take the problem from being O(2^N) slower than the SGI proposal to being O(2*N) slower (in the above example, scripts B and C would still be executed twice). A 2 times slowdown for many simple scripting situations is unacceptable.

TimerEvent

The MSS proposal has a very discrete model of time. A TimerEvent may be added to get a method called every N seconds (where N is a floating point number). Like the CULL_IN/etc events, a well-behaved script will have to be written carefully so that it is not constantly being executed because of TimerEvents going off; the script will have to remove the TimerEvent when stopped, and add the TimerEvent when re-started.

Time is continuous in the SGI proposal; time-driven objects are (conceptually) evaluated constantly. The fact that this isn't possible in practice and time must be discretely sampled is an (important) implementation detail that VRML authors will not, in general, need to be aware of. The SGI proposal's TimeSensor (roughly equivalent to the TimerEvent) has explicit start and stop times, and all user-input is tagged with the time that the user input event occured. Forcing authors to deal with time as discrete events will hurt performance (since authors will explictly have to register/unregister TimerEvents to control when the events start/stop), and will cause problems as VRML is integrated with truly continuous media such as sound.

Re-usability

We believe that the SGI approach will result in object/behavior components that are more re-usable, for the following reasons:

Compiled-in connections

In the MSS proposal, a script explicitly names the nodes and fields which it sets and gets. This approach make the script not usable with nodes and fields that have different names, i.e. the script cannot be reused. Thus, the author is forced to write a new script just because they want to modify the translation and rotation fields of a Transform named "Foo" when they have a perfectly good script that modifies the translation and rotation fields of a Transform named "Bar". Connections in our proposal permit us to make this binding at authoring time (and not at the script writing time).

The "actionOn" field appears to be an attempt to address this, but will fail as soon as a script needs to modify more than one object (e.g. a script modifies the "actionOn" object and the object named "Bar", and I'd like to re-use it to act on two other, different objects).

Fragile name problem

When you rename a node, you must change all scripts that refers to that node otherwise you will encounter an error when you try to run the script. With connections, you still have to change the connections but you can detect dangling connections at load time rather than at run time.

We believe it is best to detect problems as early as possible, and believe that explicit connections in the scene will allow authoring programs to bring problems to the attention of the VRML author. If the connections are "hidden" inside scripts, the author will not know there is a problem until the script is run.

Access control: anonymous nodes

Anonymous nodes (nodes with no name) gives the author a certain minimal amount of access control; you can only make changes as specified by the connections, but nothing else. Because the only way to change anything in the MSS proposal is by looking it up by name, if you can change part of a node you can change all of it.

Persistence

The MSS proposal does not address the persistence requirements that users will require VRML browsers to support. As VRML worlds get large, interactive, and interesting, it will be important for users to be able to "save the state of the world". For example, imagine a single-user VRML adventure game (something like Myst, perhaps) that takes a long time to solve. The user will have to be able to save where they are in the puzzle to come back later.

In the SGI proposal, we assume that "the state of the world" is available to the VRML browser-- that the browser has enough information to determine what has changed, and when the user quits the browser or saves a "bookmark" to come back to later, the browser will be able to do something like:
Remember the URL for where the world was fetched
Remember the current view
Store a list of changes that the user has made to the base world

When the user re-starts or goes back to the saved bookmark, the browser will:
Fetch the 'pristine' world from the URL
Apply the changes that the user made to the world
Set the current view

Note that a similar process must also occur if the world is very large and the browser needs to unload part of the world to make room for some other part; any changes to the part of the world being unloaded will have to be remembered to maintain the illusion of a continuous, very large virtual world.

In the MSS proposal, not enough information is available to the browser to properly save and restore the state of the world--this could be rectified by either requiring script authors to implement save/restore methods, or by adding fields/variables that the script can read/write and that the browser is aware of. However, it isn't clear exactly where those fields/variables would be stored, since scripts are associated with Separators (the script is not a separate object like SGI's Logic node).

It is possible to write Logic nodes using the SGI proposal which also have this problem; however, the SGI proposal will make it much more natural for authors to create well-behaved Logic nodes that can be correctly saved/restored by browsers.

Overloading the Scene Hierarchy

The MSS proposal associates scripts with Separators in the scene graph. We believe that overloading the meaning of the scene hierarchy in this way will make it more difficult to create worlds with behavior.

In the SGI proposal, the position of a Logic node in the scene hierarchy is irrelevant (in fact, Logic nodes that "hang off the side" of the scene hierarchy work perfectly well). There are often situations in which it doesn't make sense for a Logic node to be any particular place in the scene hierarchy.

For example, imagine a script that detects when three toggle button objects are all turned on at the same time; when they are, the script turns on a light, opens a door, and starts a sound. In this example, there is no single object with which the script should be associated; putting the script on any of the three buttons, the light, the door, or the sound would be wrong.

Key differences

Given these issues, we identify several key issues that we think distill the differences between the two proposals:

1. Logic/scripts as a separate node versus logic/scripts associated with Separators. Because it is sometimes not possible to identify a particular Separator with which a script should be associated, we believe it is better to have Logic/scripts be distinct nodes.
2. Sensors as nodes or sensors done as messages/callbacks/events. We believe that using the same mechanisms for everything results in a cleaner, smaller, easier to understand and use system. The event/callback model adds extra complexity (note that the MSS proposal already has the equivalent of the SGI's connection proposal, hidden as the "tiny language" of the "actionMethod" field).
3. Explicit connections versus implicit name lookup. We would like the get/set paths between objects to be known to the browser at read time, so that the browser can intelligently optimize and re-organize the scene graph for maximum efficiency, both for objects that it knows might change and for objects it knows will never change. We allow arbitrary changes to any or all of the scene graph, but require world creators to explictly state which parts of the scene a script will be editing (a script writing into a NodeReference that is inserted at a particular place in the world). Because the MSS proposal allows any script to look up arbitrary (either named or prototyped) nodes, browsers will have to assume that there is an implicit connection between every script and every named or prototyped node.

Many of the issues raised in this document are subtle, and become apparent only after a system has been implemented and used. They represent hard-won knowledge gained by the design and implementation of 3D products such as Open Inventor, WebSpace Navigator, WebSpace Author, IRIS Inventor, IRIS Performer and IRIS Annotator.