A Global Routing Mechanism for Modular VRML-Worlds

S. Mader

Department Visualization and Interaction Techniques,
Fraunhofer-IGD, Division Rostock, Germany

Abstract

With the increasing complexity of virtual worlds VRML97 reveals its lack of native support for logical modularization of these worlds. Though the inherent mechanisms as `PROTO`*-typing and Inlining allow us to partition our worlds into several files, the restriction to file-limited namespaces makes an event-routing for complex interactions between different modules a rather difficult task. In this paper we present an easy-to-use method for event-based communication across modular VRML scenes. Based on VRML´s event model, this solution goes conform with the VRML97 standard ISO/IEC 14772-1:1997*[1]*. In order to show its ability to support the implementation of highly interactive and complex VRML-worlds we demonstrate its application in a current project from the fields of virtual heritage.*

1. Motivation

The Virtual Dunhuang Art Cave project[2, 3, 4] is a cooperation project between Zhejiang University, Hangzhou, China and the Fraunhofer Institute for Computer Graphics, Germany. It focuses on the development of a virtual heritage system supporting the preservation, restoration, promotion and replication of Dunhuang Art. One of the early results of this project is the Dunhuang InfoWeb[5], a web-based information system which presents Dunhuang Art within a multimedia framework. Essential elements of this system are 3-dimensional models of selected caves implemented using VRML97.

In its final stage the information system is intended to feature about 25 to 30 models from a total of 570 Dunhuang caves. Navigation, interaction and presentation schemes will be similar for all models. Due to this redundancy, monolithical implementations introduce a considerable transmission overhead, loading the same schemes again for each cave model. The separation of those common schemes from cave-specific content as geometry, viewpoints and lighting into different logical modules obviously yields several advantages:

reusability at runtime: Commonly used modules need to be transmitted only once. Their cached instances can be used to render subsequent cave models.

reusability at authoring stage: Once implemented, modules can be inlined a number of times.

efficient implementation and maintenance: Easy extension for additional cave models or interaction tasks. The logical structure allows the splitting of authoring tasks into well-defined subtasks.

According to the VRML97 specification[1] there are two different mechanisms of dividing contents into separate files — the use of `Inline{}`–nodes and the `PROTO`–/`EXTERNPROTO`–mechanism. Whereas the standard inlining does not provide access from the external scene to the inlined content, PROTO–types allow the definition of generic interfaces to their content. Interaction with the content of prototypes can be achieved by routing typed events to and from suitable event-slots of the interface.

However, since VRML97 does not support global name spaces, components situated in different prototypes or files cannot talk directly to each other. Each of the transmitted events requires its own declaration within the interfaces of the communicating modules and its routing through the depths of the module hierarchy. The interfaces and routing sections are growing rapidly and so does the effort for authoring and maintenance. Chris Marrin[6] also reported some performance problems with PROTO–types which originate in implementation issues. Depending on the depth of the module hierarchy there are additional costs for the routing of events. Therefore, this traditional method is very inconvenient for large, highly interactive worlds. In order to enable rich interaction between a number of modules, our goal was

Published by the Eurographics Association, ISSN 1017-4656.

to find a more efficient way to define inter-modular communication.

2. Conceptual View

There are several approaches focussing on communication issues within shared virtual environments, like the IEEE DIS standard (Distributed Interactive Simulation)[11, 12] originated in the military sector, the DWTP (Distributed Worlds Transfer and communication Protocol)[7, 8] and the VRTP (Virtual Reality Transfer Protocol)[10]. Since it is the primary objective of these approaches to support the communication between and consistency among several distributed instances of a virtual environment, all these approaches rely on complex network protocols. Besides remote communication, they also enable local communication across a range of separate modules but at the costs of high authoring effort. However, our work is focussed on efficient local communication between different modules of a single world instance.

Usually the interaction between VRML entities relies on event-based control (we use the terms 'event-based' and 'field-based' according to Brutzman[9]). Since entity names are limited to file- and PROTO-type boundaries, event-based control has some disadvantages mentioned earlier in this paper. Field-based control, however, breaks those limits. Via SFNode- or MFNode fields referenced VRML-entities are accessable 'worldwide'. According to section 4.12.9 of the VRML spec[1], script nodes are able to access event and field attributes of the referenced nodes. This, together with a modified inlining mechanism enabling the distribution of node references across file boundaries forms the basis of our approach.

The implemented framework consists of five basic components: `Router{}`, `Module{}` and `Inline{}` as well as `Source{}`- and `Target{}`-nodes for each type of VRML97–event. Each of these components is implemented as a VRML97 PROTO-type. Whereas `Module{}`- and `Inline{}`-nodes form an interactive hierarchy across multiple files (see section 3), the `Router` establishes the appropriate connections between `Sources` and `Targets`. `Sources` and `Targets` themselves define the endpoints of communication.

On world startup, a reference to the `Router` is propagated down the interactive hierarchy which the `Router` belongs to. This is done by the `Modules` and `Targets` embodying the hierarchy. Whereas `Modules` propagate the `Router` to all their components, `Inlines` just promote its reference to the first node that is found in the inlined code. This allows the definition of multiple interactive hierarchies by placing more than one `Module` in that inlined file, which is important for multiple inlining of the same file as shown in figure 1.

In return to the propagation step all interactive entities register to the `Router`. Since entity names are limited to file– and PROTO–type boundaries the routing cannot be defined by the names of start and end points. Instead, we use labels to denote the exchanged events. `Sources` and `Targets` are then interconnected by means of those labels. According to their given label the interaction targets are grouped into labeled target sets. Interaction sources gain access to those sets by obtaining an SFNode reference to the set matching their label, respectively. This all is done during the initialization of the virtual world.

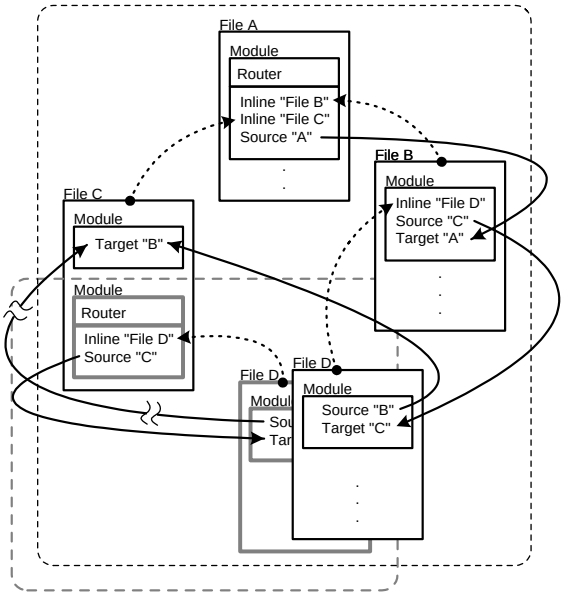

Figure 1: *Example of interactive hierarchies. Note, that in File C a second hierarchy is established by the other pair of Module and Router. Thus, there is no route between Source "B" and Target "B"!*

At runtime, the actual event routing works as follows: As soon as an event is routed to the eventIn–slot of a `Source` component, it invokes the same event at the eventOut–slots of the associated `Targets` via its reference to the target set, and thus the `Targets` themselves (field-based control). From there the events are routed to their final recipients via event-based control.

3. Users' View

As written in the above section, the implementation of our method consists of five basic PROTO's. In order to benefit from cross-module interaction one has to establish one or more interaction hierarchies. Each hierarchy must include one and only one `Router{}`–node. The simplest possible hierarchy consists of a `Router{}`, `Module{}`, `Source{}` and `Target{}` arranged like this:

```
Module {
     router Router {}
```

```
     components [
          DEF Source Source {
               label "switch_the_light"
          }
          DEF Target Target {
               label "switch_the_light"
          }
     ]
}
DEF PL PointLight {}
DEF TS TouchSensor {}

ROUTE TS.isActive TO Source.boolIn
ROUTE Target.boolOut TO PL.set_on
```

Defining additional `Module`- and `Inline`-nodes inside the components list increases the depth of the hierarchy and enables the interaction across file boundaries (see figure 1). Inlined files are ordinary VRML-files with a `Module` defined as the first top-level node. Subsequent `Modules` do not contribute to the same interaction hierarchy, and, if given their own `Router`, establish additional interaction hierarchies. Events transmitted by different interaction hierarchies do not interfere, even if they are labeled the same. This allows multiple inlining of the same file (thus, introducing the same labels!) like it is shown in figure 1.

Inserting a new interaction line simply results in the definition of a `Source{}` at the origin of the event and a `Target{}` with matching labels at the desired destination (both have to be placed inside the component list of the respective `Module{}`).

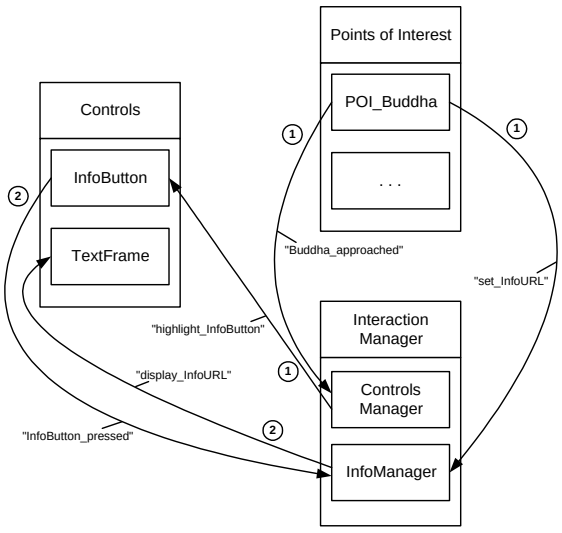

Figure 2: *Application example from the Dunhuang Info Web. The given numbers denote two different interaction lines.*

4. Results

In order to reduce the authoring and maintenance effort, and to enable both runtime and authoring reusability of modules while achieving a high level of interactivity, we applied this solution within the 'Virtual Dunhuang Art Cave' project. The logical structure of the VRML-worlds is shown in figure 3. The only modules which differ from cave to cave are the components of the 'Scene' module, containing cave related content. The complete 'Visitor' section, which makes up about 70% of the code of the whole model, is reused for all cave models. All major interaction relies on the introduced solution. We defined about 20 to 25 labeled interaction lines per cave model. An example for two interaction lines is given in figure 2.

The setup of the global routing produces some computational overhead at initialization time. At this stage, we did not yet measure the initialization delay for worlds with hundreds or thousands of interaction lines. In the case of Dunhuang Info Web a delay is not noticeable since the cave models are richly textured worlds and it takes more time to load the textures than to do the initialization. The actual event routing causes three times the costs of the ordinary (local) routing since a single interaction line usually consists of three ordinary routes: from the sensor to the `Source`, from `Source` to `Target`, and from `Target` to the actor. A comparison with the traditional PROTO–type method and its shortcommings as described in section 1 puts this additional costs into perspective again. However, an initialization delay within the range of a second and slightly higher routing costs, which do not depend on the depth of the module hierarchy, are acceptable trade-offs against the simplified authoring process and the possibility of complex interaction.

5. Future Work

At this stage the PROTO–types are implemented with the use of ECMA-script (i.e. JavaScript, see VRML–spec[1]) only. The next step will be an implementation using the VRML–JSAI (Java Script Authoring Interface, see again VRML–spec[1]). Detailed evaluation of applicability and performance will take place in the further course of Virtual Dunhuang Art Cave project. We especially intend to use this method for the implementation of the WWW–interface of tools for virtual restoration and simulation.

6. Acknowledgements

The 'Virtual Dunhuang Art Cave' project is funded by the International Bureau of DLR. We would like to thank Dr. Koepke and Mrs. Hongsernant for their support.

References

1. VRML97, the Virtual Reality Modeling Language, ISO/IEC International Standard 14772-1:1997. *http://www.vrml.org/Specifications/VRML97/.*

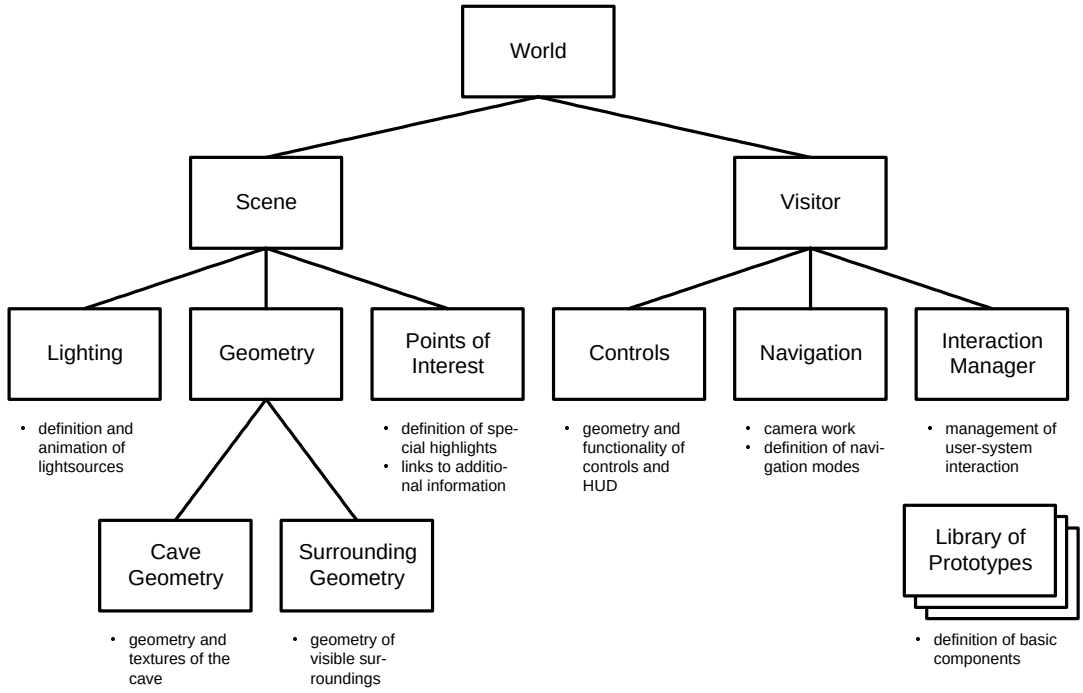

Figure 3: *Definition of modular hierarchy of the cave models.*

2. Z. Gong, D. Lu, Y. Pan Dunhuang Art Cave Presentation and preserve based on VE. *Proceedings of the Virtual Systems and MultiMedia '98*, Gifu, Japan, Nov. 1998.
3. D. Lu, X. Li,B. Wei, Y. Pan Color Restoration Techniques for Faded Murals of Mogao Grotto. *Proceedings of the Virtual Systems and MultiMedia '98*, Gifu, Japan, Nov. 1998.
4. B. Lutz, M. Weintke. Virtual Dunhuang ArtCave: A Cave within a CAVE. To appear in *Proceedings of the Eurographics'99*.
5. S. Hambach. How to Visit Dunhuang without Travelling to Central Asia. submitted to *Eurographics'99*.
6. C. Marrin. Beyond VRML — A White Paper. *http://www.marrin.com/vrml/private/EmmaWhitePaper.htm*.
7. W. Broll. DTWP – An Internet Protocol for Shared Virtual Environments. *Proceedings of the VRML'98 Symposium*, Monterey, Ca., February 16-19, 1998. ACM.
8. W. Broll. SmallTool – a ToolKit for Realizing Shared Virtual Environments on the Internet. *Distributed Systems Engineering Journal*, Vol. 5/1998, pp. 118-128, British Computer Society, The Institute of Electrical Engineers and IP Publishing, 1998.
9. D. Brutzman. The Virtual Reality Modeling Language and Java. *Communications of the ACM*, Vol. 41 no. 6,June 1998, pp. 57-64, *http://www.web3d.org/WorkingGroups/vrtp/docs/vrmljava.pdf*.
10. D. Brutzman, M. Zyda, K. Watsen, and M. Macedonia. Virtual Reality Transfer Protocol (vrtp) Design Rationale. *Proceedings Sixth IEEE Workshop on Enabling Technologies:Infrastructure for Collaborative Enterprises*, MIT Cambridge, Ma.,June 1997, pp. 179-186.
11. J. Locke. An Introduction to the Internet Networking Environment and SIMNET/DIS. *http://www-npsnet.cs.nps.navy.mil/npsnet/publications/DISIntro.ps.Z*.
12. Distributed Interactive Simulation, DIS-Java-VRML Working Group. *http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/*.