NISTIR 7322

Architecture of the Focus 3D Telemodeling Tool

Arthur F. Griesser, Ph.D.

NIST

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7322

Architecture of the Focus 3D Telemodeling Tool

Arthur F. Griesser, Ph.D.
Semiconductor Electronics Devision
Electronics and Electrical Engineering Laboratory

August 2005

Architecture of the Focus 3D Telemodeling Tool

Abstract

A high level design of the Focus 3D modeling tool is presented. Infrastructure components are selected in the light of the functional and non functional requirements of the Focus project. A description of how these components will fit together is provided, and architectural alternatives are considered.

Table of Contents

Overview

The goal of the Focus project is to improve the quality of information transfer standards by providing a tool which facilitates the collaborative construction of underlying information models. Focus will make it possible for standards participants all over the world to join together in a virtual space to build concrete models of information.

Please see the Analysis Catalog for the Focus 3D Telemodeling Tool for a more detailed description of the Focus project, an analysis class diagram, and a listing of use cases. Please also see the detailed use cases for further information on the functionality the system provides.

This high level design will discuss individually the various components that will be needed to build the Focus 3D modeling system and how these components will fit together. Each section describes the implementations considered for one component. Sections differ in their depth of coverage because components differ according to their relative importance in this project, the quality of existing implementations, and the risk perceived in using existing implementations. The final section describes, with the help of a UML component diagram, how these components interact.

Frameworks

Several virtual reality frameworks were considered as the foundation for Focus.

- Croquet: http://www.opencroquet.org/
- Studierstube: http://www.studierstube.org/
- Tinmith: http://www.tinmith.net/
- Game engines

Unfortunately none of these proved to satisfy the requirements of Focus. Croquet suffered from relatively poor performance, had problems with firewalls, and lacked a tracking component. Studierstube seemed to provide everything we needed; if we could have read the German documentation, and gotten it to run on a unix-based platform, it might have been ideal. Tinmith seemed to be better suited to augmented reality than augmented virtuality. The game engines either had licensing issues, or were not quite ready for prime time. Homunculus Flatland[1] and FreeWRL[2] were not investigated because they appear to be less complete.

[1] http://www.hpc.unm.edu/homunculus/

[2] http://freewrl.sourceforge.net/

Change Distribution

When one user makes a change to a model, the other users need to see this change in their virtual environments; somehow these changes need to be shipped around the network. Initially Distributed Open Inventor (DIV) was considered for change distribution. It proved to be difficult enough to get working on OS X (the Macintosh operating system) that it was deemed easier to rewrite this functionality than to fix DIV. Since communications can easily make or break a distributed application, the additional control over this component obtained by implementing it ourselves is desirable anyway.

The first consideration is the choice of network protocol. The three main choices[3] are:

- *Unicast* sends information to a specific computer. The sender must separately transmit to each participant's computer.

- *Broadcast* sends information to all computers on the sender's Local Area Network (LAN). The sender transmits to a single address, which corresponds to all computers on the LAN.

- *Multicast* sends information to all computers that have registered with the network to receive data sent to some multicast address they are interested in. The sender transmits to this single multicast address. Routers deliver the data to those computers that have registered to receive it. Unfortunately, true multicasting requires router support. MBone[4] is an additional networking layer that allows multicasting over a unicast network. The Spread Toolkit[5] is an interesting looking interface to multicast; rb_spread[6] provides a Ruby wrapper for this library. The streaming component of Helix[7] might also be useful.

As far as Focus is concerned, broadcast is virtually useless. Multicast appears to be ideal, but we have little experience with it. Due to its lower bandwidth requirements, multicast will be required when Focus supports audio and facial video. Model change distribution has much less stringent bandwidth requirements. Let's consider one scene graph change: a single scene element is moved and rotated. Each position requires three spatial coordinates and three orientation coordinates. If each coordinate is represented by a 64 bit IEEE double precision floating point number, that adds up to 384 bits per position. Another 16 bits should suffice to specify which element is being moved, for a total of 400 bits. To avoid the perception of flickering, the effective frame rate must approach the human "flicker fusion threshold" of about 75 frames per second (fps). A typical motion picture has a frame rate of 24 fps, which is then interrupted 2 or 3 times per frame to result in 48 or 72 effective fps. We should be able to implement the same trick, so the motion of one element would require 24 * 400 = 9,600 bits per second (of course if we can increase the frame rate to 72, so much the better). A 10 Mbps Ethernet connection

[3] http://www.tldp.org/HOWTO/Multicast-HOWTO.html

[4] http://www.savetz.com/mbone/

[5] http://www.spread.org/

[6] http://www.omniti.com/~george/rb_spread/

[7] https://helixcommunity.org/

should be able to support at least 250 moving elements, presuming less than 75 percent of the bandwidth is wasted by packet overhead and collisions. If an avatar can be adequately represented by a head, torso, elbow, and hand, an avatar moving a model element consists of five moving scene graph elements. A 10 Mbps network should therefore support scene graph updates for 50 simultaneous modelers if unicast is used. That's ample, since, if the modelers were meeting physically, typically only one or two at a time would be modifying the model.

Having selected unicast for scene graph changes, we need to decide between Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) change messages. UDP is considerably faster, at the expense of reliability. If absolute positions and angles are transmitted for each change, instead of deltas, it would not make much of a difference if UDP dropped an occasional change; the motion would be slightly jerkier. Absolute coordinates were presumed in the above bandwidth computation. Unfortunately, higher level tools often do a poor job of supporting UDP. We will therefore start out with a TCP based implementation, with the option of falling back to UDP in the unlikely event that performance becomes enough of an issue.

There are at least two topologies for change distribution:

- Each computer sends its own changes to a hub-like server that distributes the changes to all the other computers.

- Each computer sends its own changes to all other computers; communications are peer-to-peer, also called P2P. Of course a central server could still be used to register the peers participating in the exchange.

Presuming N users, each transmitting B bits worth of changes per second, the central server reduces the total (input plus output) bandwidth handled by each computer from 2(N-1)B down to NB. If much of the network stack is handled by the CPU rather than the Network Interface Card (a guess that we have not substantiated), the server would nearly halve the processor overhead required for networking; this would free up the CPU for rendering. Of course this is at the expense of the server, which has NNB worth of total traffic, resulting from NB input and N(N-1)B output. Equating the maximum output bandwidth in the P2P case (MB, where M=50-1=49) with the maximum server output bandwidth N(N-1)B and solving for N, we find that a 10 Mbps server would limit the system to about 7 users. Likewise, a 100 Mbps server could support about 22 users. The number of users should scale linearly with the number of NICs (network interface cards) the server can dedicate to sending data, until the switches feeding the server reach their capacity, or CPU overhead becomes a problem. With a dedicated server, CPU overhead probably wouldn't be an issue. Actually it is likely that neither strategy will be necessary. Standards are usually composed by surprisingly tiny teams. Even in the case of large teams, modeling will probably be performed (at least initially) by smaller subsets. The main disadvantage of the server is that latency is doubled by the simplest server implementation, which buffers the entire change before redistribution. Latency could be reduced by starting output streams as soon as the first input data are available.

There are also two models for controlling the distribution of changes:

- Push, initiated by the computer with the changed scene graph.

- Pull, where each computer interested in receiving changes polls for them.

Pull is the familiar model used by web browsers, but the required polling can waste resources and appears inelegant compared to Push. Push can also be thought of as event-driven or publish-subscribe. When a Focus client begins participating in a modeling session, the client subscribes to scene graph changes, and the server subscribes to user tracking information used to animate the user's avatar.

Above the raw socket layer, many remote procedure call implementations are available: SOAP, XML-RPC, and CORBA are among the most famous. Focus will use the Ruby drb[8] library, which provides an elegant interface for distributed computing (including some support for UDP). Actually there is a danger drb could make client-server communication too easy. For various sociological (rather than technical) reasons, it is possible for distributed applications to lose design coherence. One of the first signs of this is loss of performance due to unnecessary network traffic resulting from developer blindness to locus of execution. This can be prevented if each locus of execution is accessed through a facade[9]. Facades will be used in Focus; they are not shown in the last section's diagram because they would reduce the diagram's clarity.

In its usual configuration, drb requires a single port to be opened though firewalls. Many firewall administrators find this unacceptable, preferring new types of traffic to run though standard ports (which arguably complicates monitoring and misrepresents the traffic). Port forwarding can redirect drb traffic through standard ports, and DrbFire[10] can eliminate the need for extra ports through client firewalls. XML-RPC and SOAP normally operate across ports that firewalls usually leave open. There are many solutions to this problem; so as long as the transport layer is plug-able, firewalls should not be a serious obstacle. Encryption of distributed changes, if present, would be part of this transport layer; it could be used to protect the server and client from replay and person-in-the-middle attacks. Because several people will be viewing the model, attacks against the model itself are probably not as serious as attempts to execute arbitrary code. Tainting[11] can help defend against arbitrary code execution. Some computer languages (such as Ruby) offer built-in support for tainting.

Change distribution could be part of sessions managed by the Session Initiation Protocol[12] (SIP), which has become very popular. It's used by many Voice Over IP (VOIP) and video conferencing products, including Ekiga, Gizmo Project, and OpenWengo. H.323[13] (used by Ekiga and XMeeting) is a potential competitor. Either of these protocols would provide for voice and facial video channels, at the expense of considerable complexity. It will be far easier to let an external tool provide voice communications, and facial video is not a high priority feature. Session management should be plug-able, to make it easy to replace an initial simple custom component by a SIP or H.323 compatible solution.

---

[8] http://raa.ruby-lang.org/project/druby

[9] http://en.wikipedia.org/wiki/Facade_pattern

[10] http://rubyforge.org/projects/drbfire

[11] http://www.cs.virginia.edu/~an7s/publications/sec2005.pdf

[12] http://www.packetizer.com/voip/sip/standards.html

[13] http://www.packetizer.com/voip/h323/standards.html

GUI Toolkit

Focus will need to interact somehow with the operating system's windows. There are lots of cross-platform GUI toolkits. The following were evaluated:

- Fox (http://www.fox-toolkit.org/)
- wxWindows (http://www.wxwindows.org/)
- QT (http://www.trolltech.com/)
- Tk (http://tcl.sourceforge.net/)

Although Fox has been used at NIST to produce RSVP, OffspringViewer, and CEM Editor, QT was selected for Focus. Fox was usable, but QT has better GUI construction tools, a supported open source binding for Coin (SoQT), and widgets that more closely resemble native widgets. The QtRuby bindings appear to be satisfactory. Also the licensing terms for the QT version 4 Windows libraries allow distribution as part of an application delivered with a public domain license. Use of native widgets might have made wxWindows attractive, but its Ruby bindings are poorly documented and incomplete. Tk has no real advantage over the other toolkits, and its GUIs do not even attempt to resemble native widgets.

Renderer

OpenGL is a well known cross-platform 3D graphics rendering engine; NIST has used it in Hydra and in the OffspringViewer. OpenGL is very procedural; rendering can be a rather complicated process. In contrast, OpenInventor[14] is a higher level open-sourced object-oriented component, built on top of OpenGL, that allows assembly of entire scenes from smaller 3D objects with rendering and other useful behaviors. Coin is a popular, well supported, open source superset of OpenInventor. It cooperates nicely with ARToolkit. One potential problem is that SoQT development is done with QT version 2. The binding is compatible with the current QT 3, but may have problems with QT 4 (which has the more appropriate license).

VTK[15] is another high level 3D graphics toolkit. It may well be even more advanced than OpenInventor, but it was not fully investigated because Coin provides the required functionality and compatibility with other components, such as ARToolkit, and because we already understand the OpenGL layer that Coin is built upon. Lastly, some of the code in VTK is covered by patents, and it is not immediately obvious just how essential this patented code would be to Focus.

[14] The Inventor Mentor : Programming Object-Oriented 3D Graphics with Open Inventor, Release 2 by Josie Wernecke, Open Inventor Architecture Group, Addison-Wesley Professional

[15] http://public.kitware.com/VTK/index.php

The Irrlicht Engine (http://irrlicht.sourceforge.net/) is an interesting layer on top of your choice of OpenGL, DirectX8, and DirectX9. We discovered it after we already made a commitment to Coin, and it is not clear that Irrlicht is really ready for production development.

Tracker

OpenTracker was considered for tracking purposes. It wraps either:

- Virtual Reality Peripheral Network (http://www.cs.unc.edu/Research/vrpn/), which can obtain information from about ten different tracking devices, or

- ARToolkit (http://www.hitl.washington.edu/artoolkit/), which tracks fiducials (graphical marking symbols) seen by a camera.

Since a wrapper provides relatively little value, and Open Tracker proved difficult to get working, the Focus project will communicate directly with tracking devices. Initially all tracking will be done with ARToolkit. Head tracking will be done by calculation of the head's position and orientation with respect to a fixed fiducial. Hand tracking will be done by calculation of position and orientation of a fiducial attached to a glove. Finger positions will be measured by a P5 Glove[16], which has open source drivers for unix and Windows. The eMagin Z800 3D visor[17] contains a built in head tracking unit. An attempt will be made to interface to this head tracking unit. This product is not yet on the market, so driver compatibility has not yet been determined. The P5 Glove contains a hand position tracker which might turn out to be of use, but initial experiments suggest that its range of motion is too limited.

Persistence

Models need to be saved between sessions. A shared repository could provide version control as well as persistence. There are many choices for repository components. The main categories are relational databases (such as MySQL[18], PostgreSQL[19], Berkeley DB[20], Firebird[21], and many others), hierarchical databases (now largely fallen from favor), object databases (such as GemStone/S[22]), and native XML databases (such as eXist[23]).

---

16 http://www.videogamealliance.com/VGA/video_game/P5/P5_Specs.php

17 http://www.emagin.com/3dvisor/assets/eMaginz8003DvisorDS.pdf

18 http://www.mysql.com/

19 http://www.postgresql.org/

20 http://www.sleepycat.com/

21 http://firebird.sourceforge.net/

22 http://www.gemstone.com/products/smalltalk/index.php

23 http://exist.sourceforge.net/

Persistence is also required in the client, where it is used to cache the 3D description of Stereocons, Avatars, and the modeling environment (world). It should not be necessary to exchange this kind of information during a modeling session. Instead, when a client joins a modeling session, the timestamps on this information should be checked and the information updated if necessary.

Because the performance of the persistence service is not expected to be crucial, and because there are so many mature well-tested fallback options, it was decided to use familiar tools without further evaluation. Focus will therefore start out with the fast and easy-to-use Madeleine[24] object database. If maintenance becomes a problem due to limited tools for inspection and versioning, Focus will be able to switch painlessly to the Active Record[25] object-relation mapper, with MySQL as a back end.

Metamodel

Several UML meta-models are available:

- Eclipse Modeling Framework (http://download.eclipse.org/tools/emf/scripts/home.php), formerly known as the "XMI Toolkit"
- Sun Netbeans Metadata Repository (MDR, http://mdr.netbeans.org/)
- Novosoft UML Library (NSUML, http://nsuml.sourceforge.net/)

Ideally, these would make it easy to provide some level of interoperability with more traditional UML tools. Unfortunately, XMI appears to be a quagmire: existing tools claiming XMI compliance are not very good at model exchange, even when diagrams are not considered important. Focus will initially use its own simpler meta-model.

Discovery

Linda[26] provides a simple mechanism for service discovery and dependency injection (a way of assembling loosely coupled objects, also known as "inversion of control"). Rinda[27] is a Ruby implementation of Linda which Focus will use. Loose coupling is one of the goals of Service Oriented Architectures, which use extensive metadata (such as WSDLs) as part of the discovery process. Focus will not use detailed metadata, only the names of interfaces (which will be taken to symbolically represent all the other metadata). This kind of dynamic assembly is certainly not required to obtain the necessary functionality, but the cost is low, and (by emphasizing interfaces) it will improve the long term maintainability.

---

[24] http://madeleine.sourceforge.net/

[25] http://ar.rubyonrails.com/

[26] Ahuja, Sudhir, Nicholas Carriero, and David Gelernter, "Linda and Friends," IEEE Computer, Aug. 1986

[27] http://www.ruby-doc.org/stdlib/libdoc/rinda/rdoc/

Gesture Interpreter

OpenInventor provides "Manipulator" scene graph nodes that the user can interact with. Focus will use some of these, but it would also be desirable to support gestures, which requires decoding hand and finger motions into commands. This could be done by parsing tokens representing the motions. Parsers are usually constructed by compiler-compilers such as Yacc, from a Backus-Naur form[28] grammar. NIST has ported to Ruby a Smalltalk implementation of a compiler-compiler, which Focus will use to generate the gesture parser. A fallback approach will be to use neural networks to recognize gestures[29].

Rule Engine

Inference engines (usually called Rule Engines) draw conclusions from presented facts. They are often used to encode business rules, which helps factor the rapidly changing, domain expert maintained, business rules out of programmer-maintained source code. Focus could benefit from rule factorization in two areas. One is in validating commands and data. The other is in the meta-model. NIST has already implemented in Ruby a forward chaining inference engine based on the Rete algorithm[30].

Command Interpreter

All changes made to the model will be brought about by the Command pattern[31], through the CommandInterpreter component. These commands will be stored on a stack by the CommandInterpreter, so that an inverse operation can be applied whenever a change needs to be undone. The inverse operation is also pushed onto the stack. When a redo is necessary, the inverse operation is popped from the stack, and the command beneath it is re-executed. Each position in the stack corresponds to a slightly different version of the model. It will be possible to label positions in the stack so that users will be able to roll back to that version. This component will be written for Focus.

---

[28] http://en.wikipedia.org/wiki/Backus-Naur_form

[29] http://www.codeproject.com/cpp/gestureapp.asp

[30] Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17-37, 1982

[31] *Design Patterns, Elements of Reusable Object-Oriented Software*, by Gamma, Helm, Johnson, and Vlissides

Glue

Somehow all of these components need to be interconnected. The selection of a programming language is a very subjective task. Some people can't imagine software development without strong typing; others find it an intolerable impediment. Although we remain interested in other languages (particularly Objective C), Focus will use the Ruby[32] language.

We have been using Ruby for about a year now with no major problems, and we have built up a considerable code base. Ruby is well documented[33,34,35], is cross-platform, and has a good selection of libraries, primarily at http://raa.ruby-lang.org/ and http://rubyforge.org/. Ruby can interoperate with Java through rjb[36] and JRuby[37], although we have not yet experimented with this capability. Ruby can also interoperate with C and C++ through Swig[38]. Swig is well documented and maintained, because it supports Perl, Python, Tcl, C#, Common Lisp, and Scheme, as well as Ruby. Our initial experiments with Swig have been reasonably successful; we have not found any show-stoppers. Swig will be necessary to produce Ruby interfaces for C and C++ libraries that do not already have Ruby interfaces. At this point, that appears to be limited to Coin, ARToolkit, and tracking hardware drivers (the P5 glove and eMagin head tracker, if it is cross platform and its headers are released).

Web

It would be convenient to deliver the entire Focus application automatically through a web browser. This would seem to require wrapping Focus for interpretation by an existing browser plugin, such as the Java or Flash plugins. Unfortunately, the current state of the art would unacceptably limit the user interface and performance. Ruby does not yet have an equivalent to Java's Applets, at least in part because Ruby delegates user interface construction to libraries such as Fox and Qt. Ruby code could be accessed from Java Applets through jrb or JRuby, probably with a large performance price. The Alph[39] Flash-Ruby bridge might come closer. Either way, large libraries (such as Coin) remain a problem. The best solution would probably be a Ruby browser plugin that includes support for multiple GUI toolkits, and Coin. That's clearly outside of the current scope of Focus.

---

[32] http://www.ruby-lang.org/en/

[33] *Programming Ruby, The Pragmatic Programmer's Guide, Second Edition*, Dave Thomas, Chad Fowler, and Andy Hunt, Pragmatic Bookshelf, 2004

[34] *Ruby In A Nutshell*, Yukihiro Matsumoto, O'Reilly, 2001

[35] *The Ruby Way*, Hal Fulton, Sams, 2001

[36] http://raa.ruby-lang.org/project/rjb/

[37] http://jruby.sourceforge.net/

[38] http://www.swig.org/

[39] http://rubyforge.org/projects/alph/ and http://richkilmer.blogs.com/ether/2004/10/alph_code_relea.html

Even though browser delivery is not yet practical for the Focus modeling use cases, it would be quite satisfactory for the management use cases, which are not nearly as demanding. There are many ways to create web applications. Two popular fast ways are OpenLaszlo[40] and Ruby on Rails[41] (also known simply as Rails, or ROR). Rails is probably a better choice for Focus, since it is Ruby based. Rails is well documented[42] and supports Ajax[43]. Other interesting Ruby web frameworks (all available on Rubyforge) include IOWA, Arrow, Nitro, Labarynth / Borges, Cerise, and SWS.

Fitting It All Together

At application startup, services register themselves with Discovery. Consumers use Discovery to find the services they need and register themselves with these services. These steps are repeated at every startup. Every Session has one View on one Model, selected from the Repository. The client SceneGraph is updated from the Server Repository (this is not shown in the diagram). The Tracker then begins tracking user motions; it uses this information to pose the AvatarModel. The AvatarModel resides on the server, because it updates the AvatarRepresentation (through the SceneGraph) for all of the Clients except the source of the tracking information. Tracking information is then fed (possibly in modified/tokenized form) into the GestureInterpreter. The GestureInterpreter parses the tracking information to discover meaningful gestures, which it associates with Command objects. These Commands are submitted to the CommandInterpreter, which validates them with the RuleEngine. Some commands (such as an avatar moving a hand which is grasping a Model element) are also fed directly from the AvatarModel to the CommandInterpreter. Other commands come from callbacks from Client Manipulators to ManipulatorControllers and from user interaction with the Dashboard. The Dashboard displays flat information that supplements the 3D interface; for example, it would have a tree listing Model components, if that is not collapsed. Just about anything could be displayed here, which is why the Display interface is shown as used by the Server itself, rather than some smaller component. If a command is invalid, it is either silently ignored or displayed in the Dashboard, depending on the severity. Valid Commands are pushed onto the CommandInterpeter's stack and executed.

Depending on the nature of Command, the Model might be updated (such as the introduction of a new model element, which then inserts an appropriate representation into the SceneGraph), the Layout might be modified for some ModelElement, the Session parameters might be updated, etc. The Command might even act on CommandInterpreter itself, if it specifies an undo or redo operation. The model resides in the server because it is shared by all the participants.

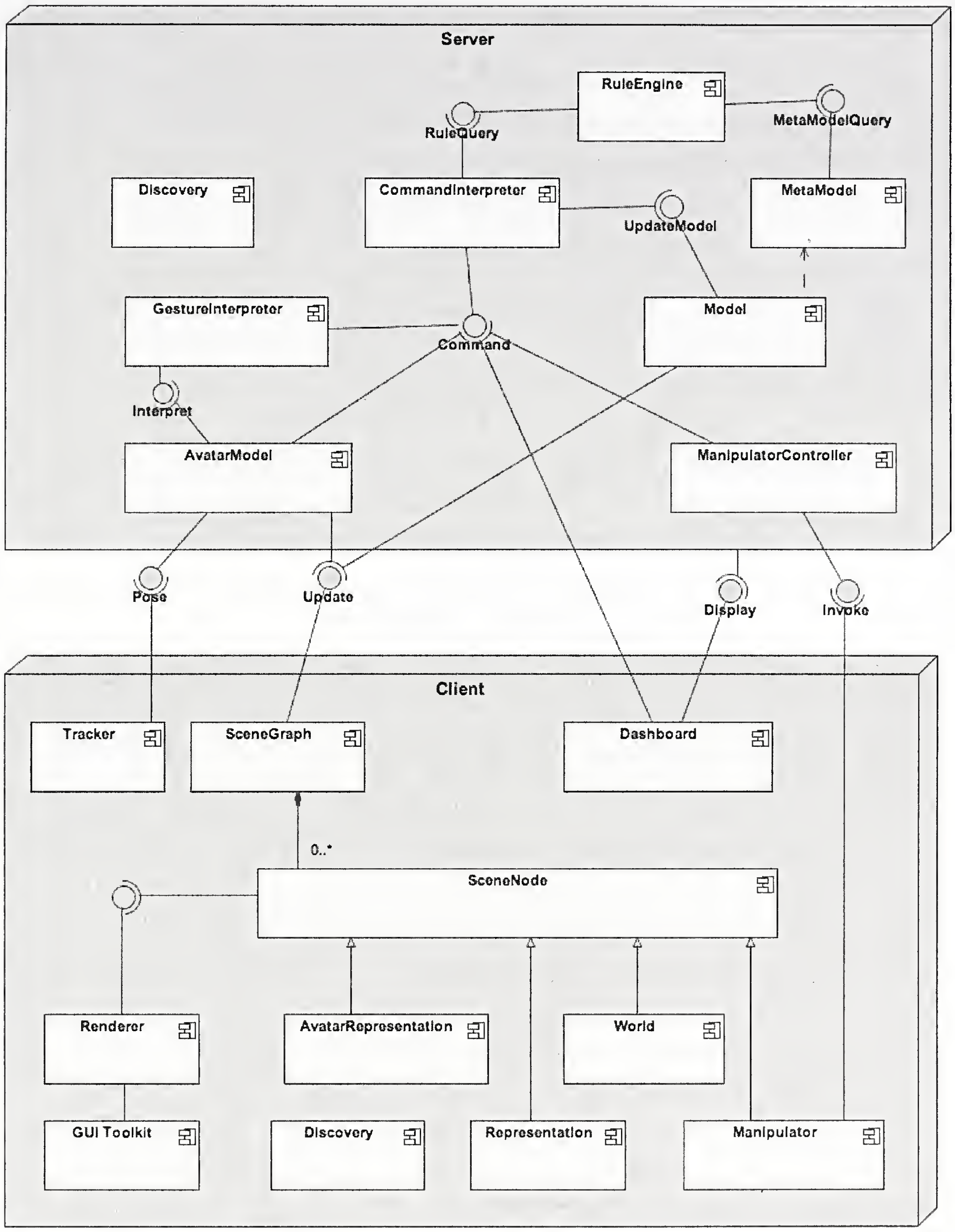
[40] http://www.laszlosystems.com/developers/

[41] http://www.rubyonrails.org/

[42] http://www.pragmaticprogrammer.com/titles/rails/index.html

[43] http://en.wikipedia.org/wiki/Ajax_%28programming%29

Figure 1: A UML Component Diagram



Revision History

Version	Date	Developer	Change
0.1	4/23/05	AFG	First cut
0.2	5/18/05	AFG	• Added brief description of Irrlicht Engine. • Improved explanation of 24 fps minimum. • Expanded 49B for P2P • Corrected description of Coin
0.3	8/27/05	AFG	• Added table of contents • Created "Overview" section. Moved the last paragraph of "Frameworks" to "Overview," and reworded • Added neural network fallback for gesture recognition • Added JRuby to section on Ruby • Added "Web" section • Modified component diagram: • Changed Undo to CommandInterpreter, to match previously articulated responsibilities • Replaced Stereocon by Representation (which includes Path and PathStyle (for Edges, such as Associations) and OrentedLocation and Stereocon (for Vertices such as Classes). • Changed AvatarView to AvatarRepresentation (for consistency) • Removed NodeCache (which is just a persistent SceneGraph) • Split Query into RuleQuery and MetaModelQuery (which is performed indirectly, through a RuleQuery) • Removed Repository and Session from diagram: they are important, but not for the purposes of this diagram. • Added client side Discovery service
0.4	3/29/06	AFG	• Expanded intro per request by Jim St. Pierre. • Added discussion of firewalls, tainting, SIP, H.323