A Modeling Strategy for the NASA Intelligent Synthesis Environment

Paul A. Fishwick

Department of Computer & Information Science & Engineering
University of Florida
Gainesville, Florida 32611, U.S.A.

July 13, 1999

Abstract

We overview the goals of NASAs Intelligent Synthesis Environment (ISE) from the perspective of *system modeling*. Some of the problems with present day modeling are discussed, followed by a suggested course of action where models as well as their objects are specified in a uniform representation based on the Virtual Reality Modeling Language (VRML). Existing dynamic modeling techniques tend to be 2D in form. The Rube methodology and application provides a 3D modeling framework where model components are objects, and all objects are defined in such a way that they can be easily defined within web documents. This approach suggests the formation of reusable *digital objects* that contain models. **Keywords: Modeling, Metaphor, Abstraction, Simulation**

1 Modeling the Future

NASA is reinventing itself with respect to new challenges that will culminate in more frequent, less expensive missions. A recent paper by Goldin, Venneri and Noor [10] covers some of the sweeping changes that are to engulf NASA and project it well into the next century. The Intelligent Synthesis Environment (ISE) represents a key piece of the new NASA. How can we begin with a conceptual design for a new spacecraft and take this design through the stages of analysis, testing and fabrication while maintaining the highest level of quality? We enable ourselves to step through sections of the gauntlet with ease if we can generate effective *modeling* methods. Modeling represents a significant part of ISE since it is with modeling that synthesis of spacecraft is made manifest. ISE is divided into five elements. While the general role of modeling is pervasive in all areas, it is strongest in the ISE elements *Rapid Synthesis and Simulation Tools* and *Collaborative Engineering Environment*. It is certainly

cheaper to build a virtual spacecraft for Cassini or the Deep Space missions than to construct the actual hardware. And yet, modeling is not without its problems. Modeling can be extraordinarily complex—both in representational schemes and in the iterative procedures required to evolve models over time. My goal is to focus on the modeling aspect of ISE to recommend specific changes in how we design dynamic models that blend seamlessly with the 3D objects being modeled. Models do not have material components. They are ethereal and live inside the computer. It is through the efficient practice of modeling that NASA will jumpstart itself into a more efficient future.

NASA Centers are embracing the goals of ISE. Kennedy Space Center [4] has the *virtual Shuttle operations model* to support ground processing. JPL is improving the approach to engineering spacecraft from design to fabrication. The Develop New Products (DNP) initiative has generated significant research in methods for improving engineering design and the processes associated with design. Smith [20] and Wall et al. [22, 21] define approaches to modifying the existing NASA engineering design practices through model-based means. They point out that much of what exists today in NASA reflects a "document-based" approach to design. A model-based approach is a significant step toward a more manageable process. A related problem is where code is used instead of models. The recognition that we need to surface models will naturally lead to more effective and cost-efficient simulations, where the code is automatically compiled, translated from the model's structure. The DNP design cycle is typically divided into the following processes: Mission and System Design (MSD), Design, Build and Test (DBAT), Validate, Integrate, Verify and Operate (VIVO) and Project Leadership and Planning (PLP). Rather than being sequential, these are concurrent and hierarchically related processes with PLP being on the top and proceeding downward to the lower levels of administrative detail as follows: $PLP \Rightarrow MSD \Rightarrow DBAT \Rightarrow VIVO$. The mission is the top-most concern of any NASA process after a project is created. The mission defines what tasks are to be done, and in what order. Sample-based missions involve the collection of material from a comet or a planet's surface. A mapping-based mission would map the surface of a planet or its satellites. Most missions are multi-facetted. For example, Cassini involves flybys and mapping of planets, a moon of Saturn as well as instrumentation for atmospheric experiments for the released Titan probe. The DNP goal is to build cross-cutting (XCUT) models that span all aspects of the mission. If a mission begins with a modeled mission and modeled spacecraft then there will be easier and more effective collaboration among designers, engineers and manufacturing staff. Off the shelf commercial software for data flow diagrams and state-based diagrams have been used recently for elaborating modeled spacecraft subsystems.

The ISE Goal is to "develop the capability for personnel at dispersed geographic locations to work together in a virtual environment, using computer simulations to model the complete life-cycle of a product/mission before commitments are made to produce physical products" [15]. This is an ambitious goal, but it is on target with the increasing use of modeling and simulation to improve the efficiency by which we design and manufacture

components, machines, aircraft carriers, process plants and spacecraft. It also builds upon existing NASA projects (i.e., DNP) that attempt to steer engineering beyond paper and documents to digital representations of objects. In short, we need to use today's cheap computer technology to manufacture virtual equivalents of what we buy and sell.

One of the problems with DNP is that it uses a centralized parameter database, around which programs are situated so that each program reads-from and writes-to the database. This central hub-spoke approach is an improvement over having separate programs each with separate data files and repositories; however, a cleaner approach is to create an object-oriented scene where all data are associated with the relevant objects. The central database approach [9] was also used for the NASA Integrated Programs for AeroSpace Vehicle Design (IPAD) Project [8]. IPAD used a relational database to store structure-based parameters to be used by CAD and Finite Element programs at NASA Langley Research Center. This was a dramatic improvement over separate data files, but it suffered from the fragmentation of connecting data and model to the encapsulating object. With the design and creation of a spacecraft, the scientists and engineers will interact and focus on the physical item—the spacecraft itself. If we can create a process where we build a completely digital spacecraft, then we will maintain this necessary collaboration among all programmers, modelers and engineers. Parameters of a high-gain antenna should be made available to the engineer who touches the antenna; the parameters need to be stored within the objects they define. Moreover, all information about the spacecraft should be so oriented so that we also get to programs and models via the digital spacecraft. The spacecraft itself becomes the primary interface for all related models, programs, and data. Higher level abstract concepts such as the *mission* can be materialized into objects that remind us of the basic mission elements.

2 Problems in Modeling

There are a number of problems that must be addressed once we begin to model. These problems are by no means intrinsic to NASA. They are general problems of the larger modeling community. Even though, a large segment of the engineering community acknowledges the importance of modeling, the overall process modeling is not without its share of defects. Significant changes need to be instrumented if we are to make modeling effective, for if it is not a truly economic enterprise, modeling and simulation will always be seen as choices of last resort, or "to be performed only when time and resources permit." Let's highlight important modeling issues:

- *Modeling Freedom:* Many types of modeling exist, with mathematical models being only one type. We need to reemphasize that models are for humans—not computers. Therefore the models must appeal to the human senses to be effective.

- *Modeling vs. Validation:* The aspects of modeling that we discuss are based on design principles. Even though models are said "to be good" when they validate physical phenomena, all models are flawed in this sense—the model shows us a window of valid behavior of an object and we use it to augment our intellect and otherwise mathematical methods. The Bohr billiard ball model of the atom is still very useful when used correctly even though we realize that billiard balls are not to be taken literally [11, 1]. Validation is separate from modeling but is to be used in conjunction with it. Modeling is what we do to understand and reason about a thing. Validation is taking a model and comparing the model's prediction with experiment.

- *Code vs. Model:* It is all too frequent that when one speaks of "a model" that one is referring to an abstract representation that bears little or no formal relation to the computer code that is supposed to *represent the model*. While code can be viewed as a model in its own right, more common model forms are based on both equational and highly-visual structures. It is essential to have *generated code* be driven from the model and so all interaction is directly through the model so that we can best forget the code, since code represents the cement whereas the model represents the multi-tiered building created from the cement.

- *Programs vs. Models:* How do computer programs and models relate? The differences between programming, as we generally learn it in Universities, and modeling reflect a gradual change in our software and hardware technologies. Computer Science and Engineering stands out as being separate from other Engineering disciplines in the sense that everyone else talks about matter and physics and computer scientists talk of data structures, procedures, relations and objects. Programming has evolved with a heavy bias toward mathematical representation. The problem is that this sort of representation bears little direct connection to physics or to other engineering disciplines. Fortunately, movements are underway in many computer science areas that suggest alternate, more physical, representational structures [17, 2, 19].

- *Integration:* NASA is in need of truly integrated virtual, 3D environments where the objects to be modeled, as well as their models, live in the same space. To determine the dynamics of the Cassini probe destined for Titan, one need only touch the 3D probe (attached to the orbiter), activate its *behavior field* object and then navigate the dynamics that are surfaced in a 3D form. Achieving this means that we have to free the process of dynamic modeling from its two-dimensional home where it has been imprisoned. Humans better understand and reason with environments that are similar to those found in every day life. Data defining parameters of spacecraft science instrumentation, for example, should be co-located with the virtual instruments. Parameters are part of objects and the engineer wants to reason and work with these parameters through the virtual objects that the data represent or modify. It may well be that a very low-level underlying database schema supporting such interaction is still

needed, but it is critical to maintain the virtual connections to the data through the physical spacecraft components. This might be seen as an issue of *visualization* or *user interface* and it is. The act of modeling is all about developing and fostering sensory appeal of the human to the modeled object. Thus, it becomes impossible to separate the discipline of human-computer interaction from the task of modeling. They are one and the same. The relational or hierarchical database should disappear from view since it bears no relation to the spacecraft.

3 The Nature of Modeling

One physical object captures some information about another object. If we think about our plastic toys, metal trains and even our sophisticated scale-based engineering models, we see a common thread: to build one object that says something about another—usually larger and more expensive—object. Let's call these objects the *source object* and the *target object*. Similar object definitions can be found in the literature of metaphors [12] and semiotics [18]. The source object *models* the target, and so, modeling represents a relation between objects. Often, the source object is termed *the model* of the target. We have been discussing scale models identified by the source and target having roughly proportional geometries. Scale-based models often suffer from the problem where changing the scale of a thing affects more than just the geometry. It also affects the fundamental laws applied at each scale. For example, the hydrodynamics of the scaled ocean model may be different than for the real ocean. Nevertheless, we can attempt to adjust for the scaling problems and proceed to understand the larger universe through a smaller, more manipulable, version.

Later on in our education, we learned that modeling has other many other forms. The mathematical model represents variables and symbols that describe or model an object. Learning may begin with algebraic equations such as $d = \frac{1}{2}at^2 + v_o t + d_0$ where d, v and a represent distance, velocity and acceleration, and where d_0 and v_0 represent initial conditions (i.e., at time zero) for starting distance and initial velocity. These models are shown to be more elegantly derived from Newton's laws, yielding ordinary differential equations of the form $f = ma$. How do these mathematical, equational models relate to the ones we first learned as children?

To answer this question, let's first consider what is being modeled. The equations capture attributes of an object that is undergoing change in space (i.e., distance), velocity and acceleration. However, none of the geometrical proportions of the target are captured in the source since the structure of the equations is invariant to the physical changes in the target. A ball can change shape during impact with the ground, but the equations do not change their *shape*. If a ball represents the target, where is the source? The source is the medium in which the equations are presented. This may, at first, seem odd but it really is no different than the toy train model versus the actual train. The paper, phosphor or blackboard—along with the

Figure 1: Painting by Rene Magritte. Is it a pipe or a *model* of a pipe?.

medium for the drawing, excitation or marking—has to exist if the equations are to exist. In a Platonic sense, we might like to think of the equations as existing in a separate, virtual, non-physical space. While one can argue their virtual existence, this representation-less and non-physical form is impractical. Without a physical representation, the equation cannot be communicated from one human to another. The fundamental purpose of representation and modeling is communication. Verbal representations (differential air pressure) are as physical as those involving printing or the exciting of a phosphor via an electron beam. Figure 1 displays a painting by the French surrealist artist Magritte, which captures the essence of *semiotics* and reminds us that source and target objects both must exist. The painting includes a phrase in French "This is not a Pipe." The object is a painting representing a pipe, or more accurately, it is a piece of paper representing a painting that, in turn, represents a pipe. In the same sense as Magritte's painting isn't a pipe, likewise, the equations are (source) objects that we interpret as attributes of other (target) objects. We see an equation and think of the target's attributes. This leaves us with the wonderful thought that when we model, regardless of the type of model, we use different objects to represent the attributes of other objects. It takes some serious practice to imagine that strange ink impressions on paper might actually represent the position of a ball, train, or horse but that is part of the wonder of modeling and of our ability to perform *abstraction*: any object can be used as a surrogate for another object's attributes. In this sense, the more abstract a source object in its relation to the target, the fewer attributes will be found to be in common: a scale model of a train preserves geometry under the right scale transformations whereas the paper and ink (representing equations) preserves none of this geometry. The equations are said to be more abstract than the scale model. There is one thing to keep in mind regarding mathematical and event 2D image-based models. We use them so frequently because of economic reasons

and not because they reflect the best and most natural ways to model. Creating a scale model of the ocean is much easier than using the real ocean. But using a piece of paper or a blackboard is even easier. What if one could create virtual 3D spaces with ease on a portable digital assistant (PDA) device? In the far future, we may even approach the environment of the Holodeck as demonstrated in Star Trek: The Next Generation. The Holodeck is a physical space where humans enter fully immersive and interactive 3D simulations. What will modeling be like in such an environment? Will we still draw things on paper or will we gesture to each other while forming 3D worlds that appear before our eyes? The ultimate goal of modeling is not that different than what we did in the sandbox. The difference is that now we can make a virtual sandbox.

4 Rube: Building the Infrastructure

Since 1989 at the University of Florida, we have constructed a number of modeling and simulation packages. We'll begin with some early packages and proceed toward our development of the Rube environment. The web will become a repository for objects as well as documents. The first package was a set of C programs called *SimPack* [6]. SimPack is a collection of C libraries and programs to allow the student to learn how to effectively simulate discrete event and continuous systems. Discrete event simulation involves irregular leaps through time, where each leap is of a different duration. Discrete event simulation requires scheduling, event list data structures, and an ability to acquire resources and to set priorities. Continuous simulation involves stepping through time using equal-sized time intervals, and is most often associated with systems based ultimately on physical laws. SimPack began as a library for discrete event handling and grew to support continuous modeling (with difference, ordinary and delay-differential equation editors). Fully interactive programs were built upon the core routines and inserted into the SimPack distribution. SimPack is widely used by a number of sites worldwide.

By the early 90s, object-oriented programming was becoming increasingly common in simulation. This suggested that we re-engineer part of SimPack to address the advantages afforded by encapsulation, class hierarchies and re-use. In 1994, we announced OOSIM. OOSIM development started with the event scheduling library in SimPack and expanded upon it to make it more robust using C++.

Both SimPack and OOSIM were found lacking in the user-interface area. Most model types used by scientists and engineers are visual. While we can encode such models in text files, the user doesn't really get a good feel for a model unless it is surfaced in a visible form. In 1997, we began development on a fully visual and interactive multimodeling system, OOPM (Object Oriented Physical Modeler) [5]. Multimodeling [7] is the practice of creating a model at one level of abstraction where each model component can be refined at a level below into a model of a different type than the one at the level above it [14]. For example,

the state components of a finite state machine can be refined into differential equations (a different model type). OOPM is based on OOSIM and has a large amount of Tcl/Tk code to support the graphical user interface (GUI). A distributed simulation executive (DSX) has also been constructed for allowing functional block model components to be distributed over the net, where each block represents a legacy code responsible for an individual simulation. This system has recently been completed. During OOPM development, we learned a number of lessons. The first lesson was that even though *multimodeling* had been explained with several formal examples, we lacked an implementation and we had to carefully work out how the scheduling of "multimodel trees" was to done. The second lesson learned was that GUI development was extremely time consuming. Although everyone wants to use a GUI, one must recognize the significant software engineering effort involved in creating a robust interface. What may appear to be very minor problems from the software engineer's viewpoint turn out to be critical errors from the standpoint of a human-computer interface. We found out that it is often better to have a primitive text-based interface that is robust than a more complex GUI that has even a very small number of user interface anomalies. Users must develop trust in an application if they are to use it with confidence.

In late 1998, we started designing Rube, named in dedication to Rube Goldberg [16], who produced many fanciful cartoon machines, all of which can be considered *models* of behavior. The procedure for creating models is as follows:

1. The user begins with an object that is to be modeled. For JPL, this can be the Cassini spacecraft with all of its main systems: propulsion, guidance, science instrumentation, power, and telecommunication. If the object is part of a larger scenario, this scenario can be defined as the top-most root object.

2. A *scene* and interactions are sketched in a story board fashion, as if creating a movie or animation. A scene is where all objects, including those modeling others, are defined within the VRML file. VRML stands for Virtual Reality Modeling Language [3], which represents the standard 3D language for the web. The Rube model browser is made available so that users can "fly though" an object to view its models without necessarily cluttering the scene with all objects. However, having some subset of the total set of models surfaced within a scene is also convenient for aesthetic reasons. The modeler may choose to build several scenes with models surfaced, or choose to view objects only through the model browser that hides all models as fields of VRML object nodes.

3. The shape and structure of all Cassini components are modeled in any modeling package that has an export facility to VRML. Most packages, such as Kinetix 3DStudioMax and Autodesk AutoCAD have this capability. Moreover, packages such as CosmoWorlds and VRCreator can be used to directly create and debug VRML content.

4. VRML PROTO (i.e., prototype) nodes are created for each object and component.

This step allows one to create *semantic attachments* so that we can define one object to be a behavioral model of another (using a *behavior* field) or to say that the Titan probe is part of the spacecraft (using a *contains* field), but a sibling of the orbiter. Without prototypes, the VRML file structure lacks semantic relations and one relies on simple grouping nodes, which are not sufficient for clearly defining how objects relate to one another.

5. Models are created for Cassini. While multiple types of models exist, we have focused on dynamic models of components, and the expression of these components in 3D. Even textually-based models that must be visualized as mathematical expressions can be expressed using the VRML text node. Models are objects in the scene that are no different structurally from pieces of Cassini—they have shape and structure. The only difference is that when an object is "modeling" another, one interprets the object's structure in a particular way, using a dynamic model template for guidance.

6. Several dynamic model templates exist. For Newell's Teapot (in Sec. 5), we used three: FBM, FSM, EQN and for Cassini (in Sec. 6), we used one: FSM. These acronyms are defined as follows: FSM = Finite State Machine; FBM = Functional Block Model; EQN = Equation Set. Equations can be algebraic, ordinary differential, or partial differential.

7. The creative modeling act is to choose a dynamic model template for some behavior for Cassini and then to pick objects that will convey the meaning of the template within the scenario. This part is a highly artistic enterprise since literally any object can be used. In VRML, one instantiates an object as a *model* by defining it: `DEF Parthenon-Complex FSM {...}`. In other words, a collection of Parthenon-type rooms are interconnected in such a way that each Parthenon-Room maps to a state of the FSM. Portals from one room to another become transitions, and state-to-state transitions become avatar movements navigating the complex.

8. There are three distinct types of roles played modelers in Rube. At the lowest level, there is the person creating the *model templates* (FSM,FBM,EQN,PETRI-NET). Each dynamic model template reflects an underlying system-theoretic model [7]. At the mid-level, the person uses an existing model template to create a *metaphor*. A Parthenon-Complex as described before is an example of an architectural metaphor. At the highest level, a person is given a set of metaphors and can choose objects from the web to create a model. These levels allow modelers to work at the levels where they are comfortable. Reusability is created since one focuses on the level of interest.

9. The simulation proceeds by the modeler creating threads of control that pass events from one VRML node to another. This can be done in one of two ways: 1) using VRML Routes, or 2) using exposed fields that are accessed from other nodes. Method 1 is familiar to VRML authors and also has the advantage that routes that extend

from one model component to an adjacent component (i.e., from one state to another or from one function to another) have a topological counterpart to the way we visualize information and control flow. The route defines the topology and data flow semantics for the simulation. Method 2 is similar to what we find in traditional object-oriented programming languages where information from one object is made available to another through an assignment statement that references outside objects and classes. In method 1, a thread that begins at the root node proceeds downward through each object that is role-playing the behavior of another. The routing thread activates Java or Javascript Script nodes that are embedded in the structures that act as models or model components for the behaviors.

10. Pre- and Post-processing is performed on the VRML file to check it for proper syntax and to aid the modeler. Pre-processing tools include wrappers (that create a single VRML file from several), decimators (that reduce the polygon count in a VRML file), and VRML parsers. The model browser mentioned earlier is a post-production tool, allowing the user to browse all physical objects to locate objects that model them. In the near future, we will extend the parser used by the browser to help semi-automate the building of script nodes.

Rube treats all models in the same way. For a clarification of this remark, consider the traditional use of the word "Modeling" as used in everyday terms. A model is something that contains attributes of a target object, which it is modeling. Whereas, equation and 2D graph-based models could be viewed as being fundamentally different from a commonsense model, Rube views them in exactly the same context: everything is an object with physical extent and modeling is a relation among objects. This unification is theoretically pleasing since it unifies what it means to "model" regardless of model type.

5 Example 1: Newell's Teapot

In the early days of computer graphics (c. 1974-75), Martin Newell rendered a unique set of Bézier surface spline patches for an ordinary teapot, which currently resides in the Computer Museum in Boston. The teapot was modeled by Jim Blinn and then rendered by Martin Newell and Ed Catmull at the University of Utah in 1974. More recently, Fish produced the image of the teapot in Fig. 2, which has the nice property of showing the internal and external teapot shape. While at this late date, the teapot may seem quaint, it has been used over the years as an icon of sorts, and more importantly as a benchmark for all variety of new techniques in rendering and modeling in computer graphics. The Teapot was recently an official emblem of the 25th anniversary of the ACM Special Interest Interest Group on Computer Graphics (SIGGRAPH).

One of our goals for Rube was to recognize that the Teapot could be used to generate

Figure 2: Newell teapot rendering by Russ Fish, Copyright © 1995, University of Utah

another potential benchmark—one that captured the entire teapot, its contents and its models. The default teapot has no behavior and has no contents; it is an elegant piece of geometry but it requires more if we are to construct a fully *digital teapot* that captures a more complete set of knowledge. In its current state, the teapot is analogous to a building façade on a Hollywood film studio backlot; it has the shape but the whole entity is missing. In VRML, using the methodology previously defined, we built TeaWorld in Fig. 3 As in Fig. 2, we have added extra props so that the teapot can be visualized, along with its behavioral model, in a reasonable contextual setting. The world is rendered in Fig. 3 using a web browser. *World* is the top-most root of the scene graph. It contains a *Clock*, *Boiling_System*, and other objects such as the desk, chairs, floor and walls. The key fields in Fig. 4 are VRML nodes of the relevant field so that the *contains* field is refers to multiple nodes for its value. This is accomplished using the VRML *MFNode* type. The hierarchical VRML scene graph for Fig. 3 is illustrated in Fig. 4. The scene contains walls, a desk, chair and a floor for context. On the desk to the left is the teapot which is filled with water. The knob controlling whether the teapot heating element (not modeled) is on or off is located in front of the teapot. To the right of the teapot, there is a pipeline with three machines, each of which appears in Fig. 3 as a semi-transparent cube. Each of these machines reflects the functional behavior of its encapsulating object: *Machine1* for *Knob*, *Machine2* for *Water* and *Machine3* for *Thermometer*. The *Thermometer* is a digital one that is positioned in *Machine3*, and is initialized to an arbitrary ambient temperature of 0° C. Inside *Machine2*,

Figure 3: Office scene with Newell Teapot, dynamic model and props.

we find a more detailed description of the behavior of the water as it changes its temperature as a result of the knob turning. The plant inside *Machine2* consists of *Tank1*, *Tank2*, *Tank3*, and four pipes that move information from one tank to the next. Inside of each tank, we find a blackboard on which is drawn a differential equation that defines the change in water temperature for that particular state. The following modeling relationships are used:

- *Pipeline* is a Functional Block Model (FBM), with three functions (i.e., machines).
- *Machine* is a function (i.e., semi-transparent cube) within an FBM.
- *Plant* is a Finite State Machine (FSM) inside of Machine 2.
- *Tank* is a state within a FSM, and represented by a red sphere.
- *Pipe* is a transition within a FSM, and represented by a green pipe with a conical point denoting direction of control flow.
- *Board* is a differential equation, represented as white text.

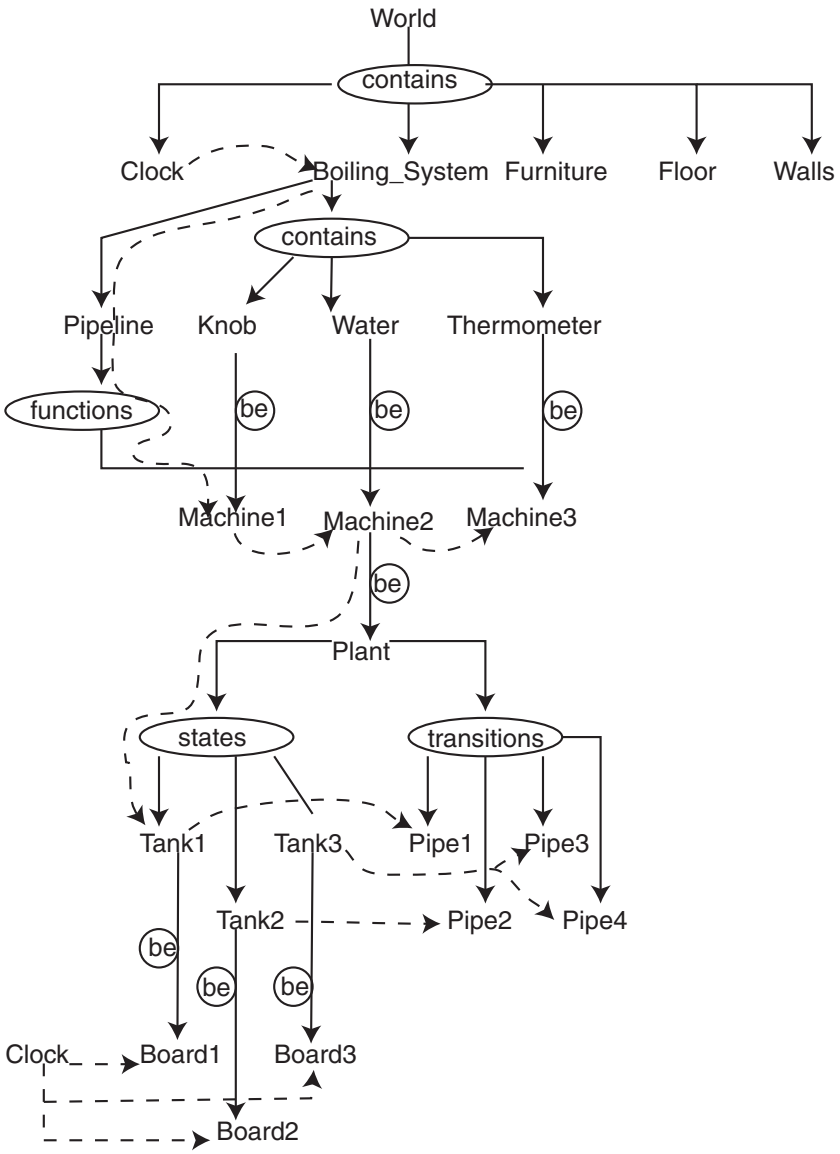


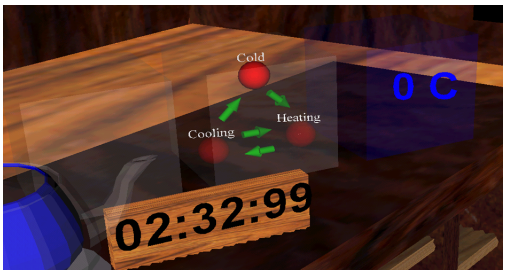
Figure 4: VRML Scene Graph for the Teapot and its models.

The following metaphors are defined in this example. The three cubes represent a sequence of machines that create a pipeline. One could have easily chosen a factory floor sequence of numerically controlled machines from the web and then used this in TeaWorld to capture the information flow. Inside the second machine, we find a plant, not unlike a petroleum plant with tanks and pipes.

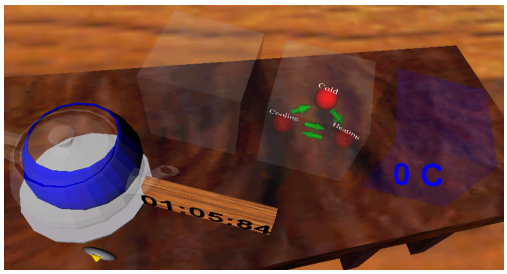
The *Pipeline* and its components represent physical objects that can be acquired from the web. For our example, we show simple objects but they have been given meaningful real-world application-oriented names to enforce the view that one object models another and that we can use the web for searching and using objects for radically different purposes than their proposed original function. The overriding concern with this exercise is to permit the modeler the freedom to choose *any* object to model *any* behavior. The challenge is to choose a set of objects that provide metaphors that are meaningful to the modeler. In many cases, it is essential that more than one individual understand the metaphorical mappings and so consensus must be reached during the process. Such consensus occurs routinely in science and in modeling when new modeling paradigms evolve. The purpose of Rube is not to dictate one model type over another, but to allow the modelers freedom in creating their own model types. In this sense, Rube can be considered a meta-level modeling methodology.

The simulation of the VRML scene shown in Fig. 4 proceeds using the dashed line thread that begins with the *Clock*. The clock has an internal time sensor that controls the VRML time. The thread corresponds closely with the routing structure built for this model. It starts at *Clock* and proceeds downward through all behavioral models. Within each behavioral model, routes exist to match the topology of the model. Therefore, *Machine1* sends information to *Machine2*, which accesses a lower level of abstraction and sends its output to *Machine3*, completing the semantics for the FBM. The FSM level contains routes from each state to its outgoing transitions.

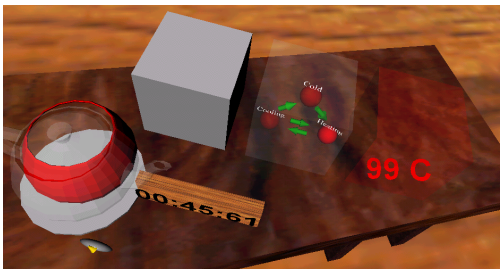
Fig. 5(a) shows a closeup view of the pipeline, that represents the dynamics of the water, beginning with the effect of the turning of the knob and ending with the thermometer that reads the water temperature. Figs. 5(b)-(d) show the pipeline during simulation when the knob is turned on and off at random times by the user. The default state is the cold state. When the knob is turned to the on position, the system moves into the heating state. When the knob is turned again back to an off position, the system moves into the cooling state and will stay there until the water reaches ambient room temperature at which time the system (through an internal state transition) returns to the cold state. Temperature change is indicated by the color of *Water* and *Machine3*, in addition to the reading on the *Thermometer* inside of *Machine3*. The material properties of *Machine1* change depending on the state of the knob. When turned off, *Machine1* is semi-transparent. When turned on, it turns opaque. Inside *Machine2*, the current state of the water is reflected by the level of intensity of each *Plant*. The current state has an increased intensity, resulting in a bright red sphere. The dynamics of temperature is indicated at two levels. At the highest level of the plant, we have a three state FSM. Within each state, we have a differential equation. The



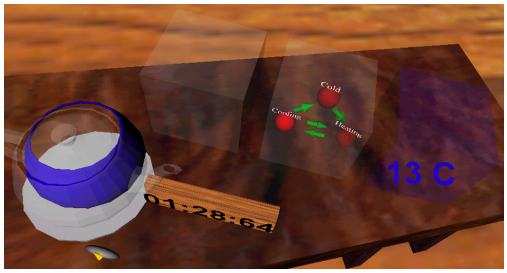
(a) Pipeline closeup.



(b) Cold State.

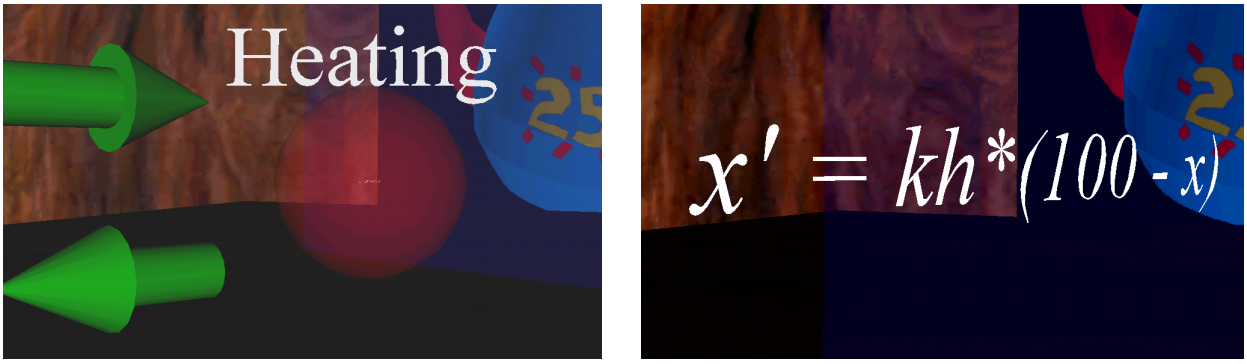


(c) Heating State.



(d) Cooling State.

Figure 5: The pipeline behavioral model and the behavioral FSM states defining the phase of the water.



(a) Outside of Heating phase.

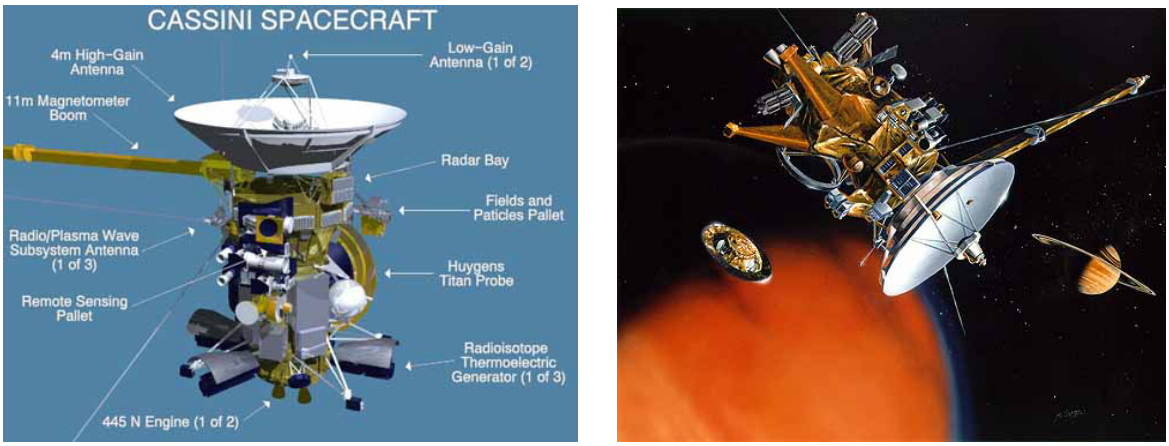
(b) Inside of Heating phase.

Figure 6: Zooming into the heating phase (Tank2).

equation is based on Newton's Law of Cooling and results in a first order exponential decay and rise that responds to the control input from the knob. The visual display of temperature change confirms this underlying dynamics since the user finds the temperature changing ever more slowly when heating to 100°C or cooling back to the ambient temperature. Figs. 6(a) and 6(b) show the outside of the heating phase (i.e., red sphere), and the inside of the phase (i.e., blackboard with the first-order differential equation).

6 Cassini

At the time of this writing (June 1999), Cassini has made a Venus flyby. It was launched in October 1997 and plans to make flybys of Venus, Earth and Jupiter on its way to Saturn. Part of the mission is to visit Titan, a moon of Saturn. Cassini, illustrated in Fig. 7(a), shows a schematic of the Cassini spacecraft while Fig. 7(b) shows an illustration of the Huygens probe separation from the Spacecraft. The probe descends through Titan's atmosphere and relays science instrument data back to the orbiter. We used the Cassini mission as a basis for a preliminary study on modeling techniques, and we decided to use an FSM dynamic model template to show three phases for the probe: 1) Separation from the spacecraft, 2) Descent, and 3) Impact. A scene was created by using an architectural metaphor for FSM states. In VRML, the user is located in a room that contains a free-floating model of Titan and Cassini. These models, as well as the model of the room, are visual, computer graphic models meant



(a) Spacecraft schematic.

(b) Release of Huygens probe.

Figure 7: Cassini mission to Saturn and Titan, Courtesy of the Jet Propulsion Laboratory.

to act as scaled-down replicas of the actual objects. Scales are non-uniform since Cassini would be much smaller with respect to Titan. The user can freely navigate this environment to view Cassini and Titan. Cassini is shown, with probe attached, making a circular orbit of the moon.

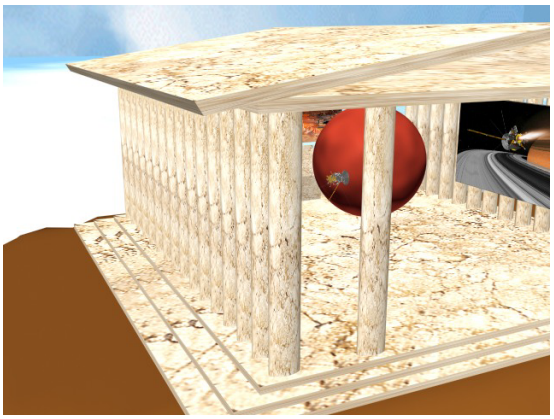
These sorts of visual, scale models are common in computer graphics but they represent a small piece of information about Cassini and its mission. Fig. 8 displays snapshots of the scene with Fig. 8(a) being the Parthenon room. On three of the four walls of this room, we find color posters relating to the mission. These posters can be clicked within the browser and the user is transported to an appropriate JPL web page identified by the poster content. Under the poster, in Fig. 8(b), we have the *Parthenon Complex*, which is an architectural metaphor for an FSM, showing the probe separation in 3 discrete phases. Fig. 8(c) shows three rooms (A, B, and C). with the following structure: $A \rightarrow B \rightarrow C$. The initial entry room and the three room environment were created from the Parthenon in Greece. This is an aesthetic aspect of this modeling practice where the modeler is free to choose any type of environment or metaphor. For Cassini, many other types of architectural metaphors come to mind, including the layout of a JPL building or the entire JPL complex (since this represents a common space well known to all JPL employees working on the Cassini project). Even within the confines of the architectural metaphor, there are an infinite number of choices. Within Room A, we may have an avatar that is positioned at the entrance to

the room (ref. Fig. 8(d)). There is also a scale model of Titan with Cassini performing the dynamics *associated with the phase* associated with Room A (i.e., probe separation from the spacecraft). Rooms B and C have similar 3D Titan models with dynamics being specified for those phases. The avatar's movement from Room $A \rightarrow B \rightarrow C$ maps directly to the dynamics of probe separation, descent and impact on Titan. The user is able to control the simulation, involving the execution of the FSM, from the main gallery or from inside the complex in Room A. Given this scenario for Cassini, there are some key issues which we should address:

- *Is it a visualization*? The work in Rube provides visualization, but models such as Cassini and Newell's Teapot demonstrate active modeling environments whose existence serves an engineering purpose and not only a post-project visualization purpose for outside visitors. This sort of modeling environment is needed from the very start of a mission—as an integral piece of the puzzle known as model design.

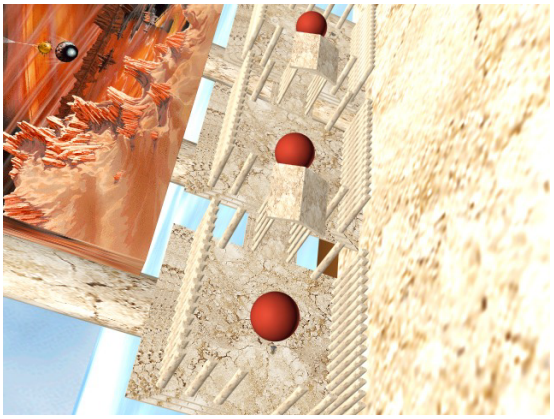
- *Is it economical?* Is this a lot of work just to create an FSM? Why go through the bother of creating the Parthenon, the complex and the avatar? All of these items are reused and so can be easily grabbed from the web. The concept of reuse is paramount to the Rube approach where the metaphor can be freely chosen and implemented. Without the web, Rube would not be possible. 3D object placement can be just as economical as 2D object placement, but object repositories are required not only for Cassini and Titan, but also for objects that serve to model the dynamic attributes of other objects (i.e., the Parthenon). Another economical aspect centers on the issue of computational speed for these models. Would creating a simulation in a more typical computer language would be more efficient? The structure of objects and their models within a VRML scene can be translated or compiled into native machine code as easily as source code; the 3D model structure becomes the "source code."

- *What is the advantage?* If we consider psychological factors, the 3D metaphor has significant advantages. First, 3D spatially-specific areas serve to improve our memory of the models (i.e., mnemonics). Second, graphical user interfaces (GUIs) have shown that a human's interaction with the computer is dramatically improved when the right metaphors are made available. Rube provides the environment for building metaphors. One should always be wary of mixed metaphors. We leave the ultimate decision to the user group as to which metaphors are effective. A Darwinian-style of evolution will likely determine which metaphors are useful and which are not. Aesthetics plays an important role here as well. If a modeler uses aesthetically appealing models and metaphors, the modeler will enjoy the work. It is a misconception to imagine that only the general populous will benefit from fully interactive 3D models. The engineers and scientist need this sort of immersion as well so that they can understand better what they are doing, and so that collaboration is made possible.

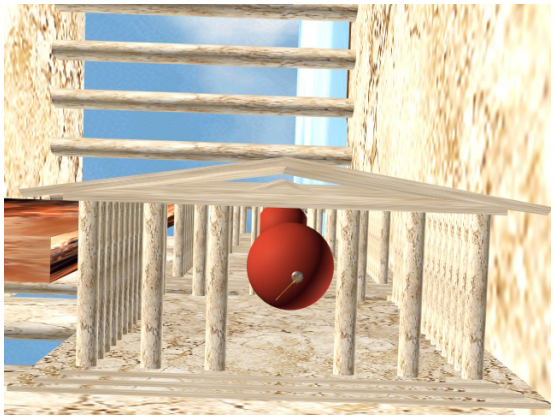


(a) View of main gallery (Parthenon room).

(b) View of the Parthenon complex.



(c) Removing the roof.



(d) Side view of complex.

Figure 8: Scene for Cassini and the Huygens probe dynamics.

- *Is this art or science?* The role of the Fine Arts in science needs strengthening. With fully immersive models, we find that we are in need of workers with hybrid engineering/art backgrounds. It is no longer sufficient to always think "in the abstract" about modeling. Effective modeling requires meaningful human interaction with 3D objects. So far, the thin veneer of a scale model has made its way into our engineering practices, but when the skin is peeled back, we find highly abstract codes and text. If the internals are to be made comprehensible (by anyone, most importantly the engineer), they must be surfaced into 3D using the powerful capabilities of metaphors [13, 12]. This doesn't mean that we will not have a low level code-base. Two-dimensional metaphors and code constructs can be mixed within the 3D worlds, just as we find them in our everyday environments with the embedding of signs. At the University of Florida, we have started a *Digital Arts and Sciences* Program with the aim to produce engineers with a more integrated background. This background will help in the production of new workers with creative modeling backgrounds.

7 Key Architectural Benefits of Rube

The following are novel features of Rube and represent reasons for choosing elements of this architecture:

- *An Integrated Environment*: There is no difference between objects modeling other objects and objects acting in their traditional roles. The modeling and object environments are identical. A pipe can be used in a petro-chemical factory or in a Petri net. Model components are chosen from the vast universe of VRML objects on the web. Components in models are dynamic as for any object. Models need not be static.

- *Modeling Freedom*: Any 2D or 3D package can be used to create models. There is no need for the Rube team to build a GUI for each model type; the model author can freely choose among drawing and modeling packages.

- *Model Design Flexibility*: There is no predefined modeling method. If a set of objects is to be interpreted as a model then one adds a small amount of "role playing" information to the objects. Any number of model types can be supported. A side-effect of this flexibility is the provision of natural *multimodeling* support.

- *VRML encapsulation*: VRML worlds can be stored anywhere over the web and positioned within an author's world through a URL. No new standards have been created outside of existing web standards and so Rube is built within the framework of VRML, but we can find expressive distributed modeling and simulation capability by "piggybacking" on the capabilities of the standard. The VRML file that contains prototypes with model fields is a *digital object*, the digital equivalent of the corresponding physical

object with all of its attributes. This encapsulation is possible due to the flexible syntax and architecture of VRML (i.e., with key nodes such as PROTO, EXTERNPROTO, Anchor nodes and Sensors being essential for the inclusion of modeling information). The average 3D file standard would leave little room for the definition of models. We propose our modeling methodology as a method for model construction with VRML. In the VMRL community, this has the potential to alter, for example, how behavior of objects are modeled. Java and selected behavior scripting languages are currently used, whereas Rube offers the capability for some of this behavior to be modeled and translated into Java using VRML, itself, to define behavior.

8 Reflections on the Art of Modeling

It is sometimes difficult to differentiate models used for the creation of pieces of art from those used with scientific purposes in mind. Models used for science are predicated on the notion that the modeling relation is unambiguously specified and made openly available to other scientists. Modeling communities generally form and evolve while stressing their metaphors. In a very general sense, natural languages have a similar evolution. The purpose of art, on the other hand, is to permit some ambiguity with the hopes of causing the viewer or listener to reflect upon the modeled world. Some of the components in worlds such as Fig. 3 could be considered non-essential modeling elements that serve to confuse the scientist. However, these elements may contribute to a more pleasing immersive environment. Should they be removed or should we add additional elements to please the eye of the beholder? In Rube, we have the freedom to go in both directions, and it isn't clear which inclusions or eliminations are appropriate since it is entirely up to the modeler or a larger modeling community. One can build an entirely two dimensional world on a blackboard using box and text objects, although this would not be in the spirit of creating immersive worlds that allow perusal of objects and their models.

It may be that a select number of modelers may find the TeaWorld room exciting and pleasing, and so is this pleasure counterproductive to the scientist or should the scientist be concerned only with the bare essentials necessary for unambiguous representation and communication? Visual models do not represent *syntactic sugar* (a term common in the Computer Science community). Instead, these models and their metaphors are essential for human understanding and comprehension. If this comprehension is complemented with a feeling of excitement about modeling, this can only be for the better. Taken to the extreme, a purely artistic piece may be one that is so couched in metaphor that the roles played by objects isn't clear. We can, therefore, imagine a kind of continuum from a completely unambiguous representation and one where the roles are not published. Between these two extremes, there is a lot of breathing space. Science can be seen as a form of consensual art where everyone tells each other what one object *means*. Agreement ensues within a community and then there is a mass convergence towards one metaphor in favor of another.

We are not proposing a modification to the VRML standard although we have found that poor authoring support currently exists in VRML editors for PROTO node creation and editing. We are suggesting a different and more more general mindset for VMRL—that it be used not only for representing the shape of objects, but all modeling information about objects. VRML should be about the complete digital object representation and not only the representation of geometry with low-level script behaviors to support animation. Fortunately, VRML contains an adequate number of features that makes this new mindset possible even though it may not be practiced on a wide scale. While a VRML file serves as the digital object, a model compiler is also required for the proper interpretation of VRML objects as models.

9 Summary

There is no unified modeling methodology, nor should there be one. Instead, modelers should be free to use and construct their own worlds that have special meaning to an individual or group. With Rube, we hope to foster that creativity without limiting a user to one or more specific metaphors. Rube has a strong tie to the World Wide Web (WWW). The web has introduced a remarkable transformation in every area of business, industry, science and engineering. It offers a way of sharing and presenting multimedia information to a world-wide set of interactive participants. Therefore any technology tied to the web's development is likely to change modeling and simulation. The tremendous interest in Java for doing simulation has taken a firm hold within the simulation field. Apart from being a good programming language, its future is intrinsically bound to the coding and interaction within a browser. VRML, and its X3D successor, represent the future of 3D immersive environments on the web. We feel that by building a modeling environment in VRML and by couching this environment within standard VRML content, that we will create a "trojan horse" for simulation modeling that allows modelers to create, share and reuse VRML files.

Our modeling approach takes a substantial departure from existing approaches in that the modeling environment and the object environment are merged seamlessly into a single environment. There isn't a difference between a circle and a house, or a sphere and a teapot. Furthermore, objects can take on any role, liberating the modeler to choose whatever metaphor that can be agreed upon by a certain community. There is no single syntax or structure for modeling. Modeling is both an art and a science; the realization that all objects can play roles takes us back to childhood. We are building Rube in the hope that by making all objects virtual that we can return to free-form modeling of every kind. Modeling in 3D can be cumbersome and can take considerable patience due to the inherent user-interface problems when working in 3D using a 2D screen interface. A short term solution to this problem is to develop a model package that is geared specifically to using one or more metaphors, making the insertion of, say, the Parthenon complex rooms a drag and drop operation. Currently, a general purpose modeling package must be used carefully

position all objects in their respective locations. A longer term solution can be found in the community of virtual interfaces. A good immersive interface will make 3D object positioning and connections a much easier task than it is today.

There are many unanswered questions concerning the Rube architecture and the affect it may have on the vast community of model authors. For example, many communities have their own internal standards for behavior representation. VHDL (Very High Level Hardware Description Language) is one such community. They have expended vast resources into the use of VHDL. Should they switch to VRML or is there a way that the two standards can relate to one another? We feel that conversion techniques between the VRML file and the other file-based standards will ameliorate the potentially harsh conditions associated with a migration of standards. Some standards such as HLA (High Level Architecture) do not include a direct provision for model specification since HLA is focused on the execution of distributed simulators and simulations regardless of how they were created and from what models they were translated. In such cases, Rube will provide a complementary technology to aid in the modeling process. UML (Unified Modeling Language) unifies select visual object-oriented formalisms for representing models of software. There is no reason why someone cannot build a complete 2D representation using a 2D modeler such as CorelDraw or AutoCAD and then construct a grammar to produce the necessary target language code segments needed for UML model execution. Therefore, Rube is a more general procedure for model translation than that provided by most metaphor-fixed visual formalisms. In this sense, the following analogy holds: Rube is to Modeling-Language-X as Yacc is to Computer-Language-Y. Rube is a general purpose model creation facility and Yacc is a compiler-compiler used to create compilers for arbitrary computer language grammars.

We will continue our research by adding to Rube and extending it to be robust. In particular, we plan on looking more closely into the problem of taking legacy code and making it available within the VRML model. This is probably best accomplished through TCP/IP and a network approach where the Java/Javascript communicates to the legacy code as a separate entity. We plan on extending the VRML parser, currently used to create the model browser, so that it can parse a 3D scene and generate the Java required for the VRML file to execute its simulation. Presently, the user must create all Script nodes. The model browser will be extended to permit various modes of locating models within objects. A "fly through" mode will take a VRML file, with all object and model prototypes, and place the models physically inside each object that it references. This new generated VRML file is then browsed in the usual fashion. Multiple scenes can be automatically generated.

Acknowledgments

My first thanks go to my students. They are making Rube and the 'virtual sandbox' come alive through their hard work and inventive ideas and solutions. In particular, I would like to

thank Kangsun Lee, Robert Cubert, Andrew Reddish, Tu Lam, and John Hopkins. I would like to thank the following agencies that have contributed towards our study of modeling and simulation, with a special thanks to the Jet Propulsion Laboratory where I visited for three weeks during June 1998: (1) Jet Propulsion Laboratory under contract 961427 *An Assessment and Design Recommendation for Object-Oriented Physical System Modeling at JPL* (John Peterson, Stephen Wall and Bill McLaughlin); (2) Rome Laboratory, Griffiss Air Force Base under contract F30602-98-C-0269 *A Web-Based Model Repository for Reusing and Sharing Physical Object Components* (Al Sisti and Steve Farr); and (3) Department of the Interior under grant 14-45-0009-1544-154 *Modeling Approaches & Empirical Studies Supporting ATLSS for the Everglades* (Don DeAngelis and Ronnie Best). We are grateful for their continued financial support.

References

[1] Ian G. Barbour. *Myths, Models and Paradigms*. Harper and Row Publishers, 1974.

[2] Grady Booch. *Object Oriented Design*. Benjamin Cummings, 1991.

[3] Rikk Carey and Gavin Bell. *The Annotated VRML 2.0 Reference Manual*. Addison-Wesley, 1997.

[4] National Research Council. Advanced Engineering Environments: Achieving the Vision, Phase I, 1999. http://www.nap.edu/catalog/.

[5] Robert M. Cubert and Paul A. Fishwick. MOOSE: An Object-Oriented Multimodeling and Simulation Application Framework. *Simulation*, 70(6):379–395, June 1998.

[6] Paul A. Fishwick. Simpack: Getting Started with Simulation Programming in C and C++. In *1992 Winter Simulation Conference*, pages 154–162, Arlington, VA, December 1992.

[7] Paul A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall, 1995.

[8] Paul A. Fishwick and Charles L. Blackburn. Managing Engineering Data Bases: The Relational Approach. *Computers in Mechanical Engineering (CIME)*, pages 8–16, January 1983.

[9] Paul A. Fishwick, Thomas R. Sutter, and Charles L. Blackburn. Prototype Integrated Design (PRIDE) System: Reference Manual, Volume 2: Schema Definition. Technical Report 172183, NASA, July 1983. NASA Contractor Report, Contract NAS1-16000.

[10] Daniel S. Goldin, Samuel L. Venneri, and Ahmed K. Noor. Beyond Incremental Change. *IEEE Computer*, 31(10):31–39, October 1998.

[11] Mary Hesse. *Models and Analogy in Science*. University of Notre Dame Press, 1966.

[12] George Lakoff. *Women, Fire and Dangerous Things: what categories reveal about the mind*. University of Chicago Press, 1987.

[13] George Lakoff and Mark Johnson. *Metaphors We Live By*. University of Chicago Press, 1980.

[14] Kangsun Lee and Paul A. Fishwick. A Methodology for Dynamic Model Abstraction. *SCS Transactions on Simulation*, 1996. Submitted August 1996.

[15] John B. Malone. Intelligent Synthesis Environment: Engineering Design in the 21st Century. Slide Presentation.

[16] Peter C. Marzio. *Rube Goldberg, His Life and Work*. Harper and Row, New York, 1973.

[17] Pierre-Alain Muller. *Instant UML*. Wrox Press, Ltd., Olton, Birmingham, England, 1997.

[18] Winfried Noth. *Handbook of Semiotics*. Indiana University Press, 1990.

[19] James Rumbaugh, Michael Blaha, William Premerlani, Eddy Frederick, and William Lorenson. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[20] David B. Smith and L. Koenig. Modeling and Project Development. In *Fifth International Workshop on Simulation for European Space Programmes - SESP '98*, November 1998.

[21] S. Wall. Reinventing the Design Process: Teams and Models. In *International Astronautical Federation Specialist Symposium on Novel Concepts for Smaller, Faster and Better Space Missions*, Redondo Beach, CA, April 1999.

[22] S. D. Wall, J. C. Baker, J. A. Krajewski, and D. B. Smith. Implementation of System Requirements Models for Space Missions. In *Eighth Annual International Symposium of the International Council on Systems Engineering*, July 26-30 1998.

Author Biography

Paul A. Fishwick is Professor of Computer and Information Science and Engineering at the University of Florida. He received the BS in Mathematics from the Pennsylvania State University, MS in Applied Science from the College of William and Mary, and PhD in Computer and Information Science from the University of Pennsylvania in 1986. He also has six years of industrial/government production and research experience working at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research) and

at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation, modeling, and animation. Dr. Fishwick is a Fellow of the Society for Computer Simulation (SCS), and a Senior Member of the IEEE Society for Systems, Man and Cybernetics. Dr. Fishwick founded the `comp.simulation` Internet news group (Simulation Digest) in 1987, which now serves over 15,000 subscribers. He has chaired workshops and conferences in the area of computer simulation, and will serve as General Chair of the 2000 Winter Simulation Conference. He was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years (1988-1990) and he is on the editorial boards of several journals including the *ACM Transactions on Modeling and Computer Simulation*, *IEEE Transactions on Systems, Man and Cybernetics*, *The Transactions of the Society for Computer Simulation*, *International Journal of Computer Simulation*, and the *Journal of Systems Engineering*. He has published over 100 refereed articles, authored one textbook *Simulation Model Design and Execution: Building Digital Worlds* (Prentice Hall, 1995), and co-edited three books.