

RUBE: An XML-Based Architecture for 3D Process Modeling and Model Fusion

Paul Fishwick

University of Florida
Gainesville, Florida, USA

ABSTRACT

Information fusion is a critical problem for science and engineering. There is a need to fuse information content specified as either *data* or *model*. We frame our work in terms of fusing dynamic and geometric models, to create an immersive environment where these models can be juxtaposed in 3D, within the same interface. The method by which this is accomplished fits well into other eXtensible Markup Language (XML) approaches to fusion in general.

The task of modeling lies at the heart of the human-computer interface, joining the human to the system under study through a variety of sensory modalities. I overview modeling as a key concern for the Defense Department and the Air Force, and then follow with a discussion of past, current, and future work. Past work began with a package with C and has progressed, in current work, to an implementation in XML. Our current work is defined within the *rube* architecture, which is detailed in subsequent papers devoted to key components. We have built *rube* as a “next generation” modeling framework using our prior software, with research opportunities in immersive 3D and tangible user interfaces.

Keywords: Modeling, Simulation, Web, Aesthetics, Personalization

1. INTRODUCTION

The Air Force, like other major defense organizations, has requirements that are addressed by the multifaceted capabilities of modeling and simulation. Let’s consider these items separately, beginning with simulation. Simulation represents the execution of a dynamic model. The High Level Architecture (HLA) is an example of a project that has a focus in execution, with simulators, individuals, and groups being distributed and yet being engaged within a common simulated region of conflict.

Modeling has a different goal than execution, since models reflect interfaces that are not all that different than the models that children create for *representing* physical phenomena outside of their immediate control. Models instrument accessibility to a target phenomenon, and they do so by appealing to analogy and metaphor. In the weakest sense, a “model” is sometimes referred-to in coding terms, as a computer program. One might speak of the model for theatre conflict or a model, a reconnaissance mission, or the operation of a surface-to-air radar-based tracking computer. While a program can be seen as being equivalent to a model, greater significance is attached when modeling is viewed as a creative activity, involving substantial graphical and audio cues. The model should be easily viewed and manipulated, with components that speak to their identifications and individual functions. While modeling for engineering and science grew from mathematical notation, the most effective modeling strategies tend to gravitate toward more visual and aural forms, engaging our senses. For instance, a flow chart or state diagram are constructions that are more in the modeling spirit, than program text. The reason for this is that sensory cues trigger powerful cognitive interface modes; we are more attuned to reasoning about a process when our senses have been fully engaged, and we have a sense of immersion within the geometric and dynamic modeling environments. Modeling is, thus, heavily dependent on visual metaphor, making models a critical interface modality.

Further author information: (Send correspondence to P. Fishwick)

P. Fishwick: E-mail: fishwick@cise.ufl.edu; Web: <http://www.cise.ufl.edu/~fishwick>

2. PRIOR WORK

For the past twelve years, we have been producing simulation tools in software to support discrete event and continuous simulation. Our work began with SimPack, which was originally a set of C source, libraries, and executables. The most significant part of SimPack was its support for discrete event simulation, with substantial segments of code for future event list handling and queuing. This latter part was recoded and re-engineered as OOSIM using C++ and object-oriented design principles. The Object-Oriented Physical Multimodeling¹ (OOPM) toolkit built upon OOSIM, to provide the user with a Tcl/Tk graphical front end for multimodeling. We created support for simulating several key types of models including finite state machines, functional block models, Petri nets, and differential equations.

While this work was substantial and novel for its time, we found ourselves swept along with the World Wide Web phenomenon like everyone else. The Web promised a different way of finding information, and a way of doing simulation using client-server mechanisms. For example, it was now possible to run a queuing simulation inside of a web browser. The possibilities inherent within client versus server side model execution helped to launch the web-based simulation initiative.^{2,3} CGI scripts are one method allowing for server side execution, and Java applets and programs enable client side execution.

What does the Web have to say about modeling? This was not clear until a few years ago when XML followed in the footsteps of the Standard Generalized Markup Language (SGML). Whereas discussions of clients and servers encourage new goals of executing models, XML speaks to a more robust way to *specify*, *present*, and *represent* model structure. It is with these goals in mind, that *rube* was born.

3. RUBE PROJECT

rube is a project that has engaged us for the past two and a half years, as a follow on to our work in multimodeling and web-based modeling and simulation. The purpose of *rube* is to facilitate dynamic multimodel construction and reuse within a 3D immersive environment. Within the context of Modeling and Simulation (M&S), this project reflects a “next generation” philosophy about modeling.⁴ The project is vast in scope, but focused upon this purpose. We began work in *rube* using the Virtual Reality Modeling Language (VRML) as our primary vehicle for specification of both geometry and dynamics. This was done by using the “Prototype” extensibility mechanism of VRML. In the past year, however, we have progressed toward XML as a way to declare model and presentation semantics. I will overview the history of the project, along with the goals, and future plans. This paper also serves as a general introduction to three other papers in this volume, each covering the details of one component of the overall architecture.

We will proceed using the architecture depicted in Figure 1, referencing each subsequent section by number as it appears in the figure. For each section, we’ll cover a description of that section and end with our current state of deployment, along with future plans for development. Fig. 1 represents the existing *rube* architecture composed of seven components, each shown as a box. Names of students responsible for developing a component are shown in italics just outside of the relevant box. The boxes, with the exception of the Program box, specify XML applications. These “applications” are languages that define content. Some languages, such as X3D defined below, are very close to the “presentation layer,” whereas others such as Multimodeling eXchange Language (MXL) are further away. This layer, or shell, defines a group of XML languages that are close to the human senses and have specified presentation, often but not necessarily visual. For example, you may feel you know how a “Sphere” (found in X3D) should be presented, but a more abstract element “State” (found in MXL) might be represented in a multitude of styles.

3.1. 3D Scene without dynamics: X3D

We begin with building scene objects from languages that are close to the visual presentation layer. This is how we want our phenomenon, to be modeled, and our model to look and sound. For *rube*, we chose to begin with 3D since 3D represents our future vision for immersive interfaces. While 3D is not without its problems, we envision that the majority of these problems are technology-based, rather than conceptual or cognitive, and as technology improves the problems will be ameliorated.

Our primary presentation language is eXtensible 3D (X3D). X3D is the successor to VRML, which became an ISO standard in 1997. X3D goes further than VRML by aligning itself with the latest Web technologies, most notably XML, and by existing within a component-based framework. X3D supports several profiles, with each profile containing a set number of components. The most compact profile is the “interchange” profile, which supports a

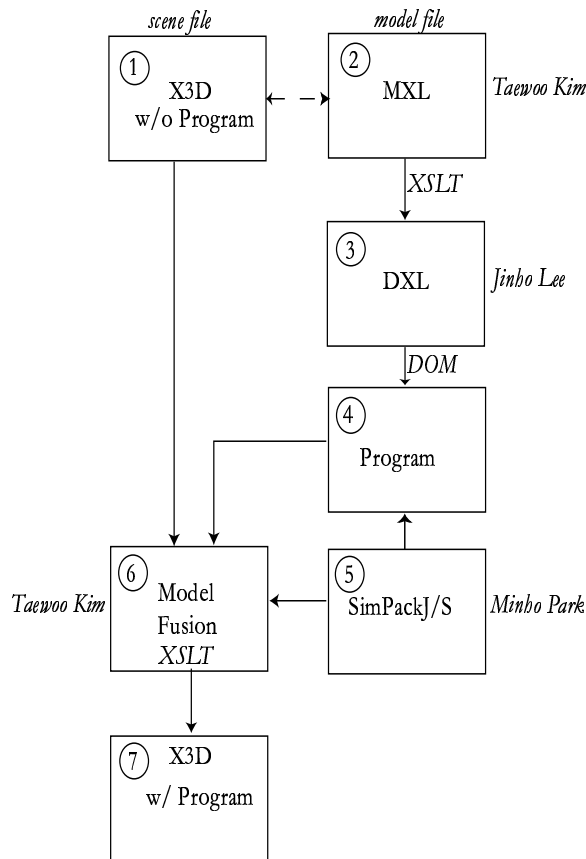


Figure 1. *rube* architecture.

minimal subset of X3D required for geometry import and export. This allows developers to begin with small implementations that may suit their requirements, without requiring all nodes from VRML. Other profiles include the VRML97 profile (compatible with VRML) and the Full Profile containing several additional useful components.

Java programs exist for the conversion of X3D to VRML and back. The open source project for X3D will read an X3D scene and use Java3D for rendering. Other X3D implementations are built upon Java, OpenGL, and DirectX. Our procedure has been to use the translators to convert from VRML to X3D, and then perform subsequent XML-based transformation and parsing using X3D. Most 3D modeling programs will export VRML, and when the final standard is presented in SIGGRAPH 2002, it is expected the these same programs will evolve to support X3D import and export. At the very least, all they need do is to incorporate existing translation software rather than to develop their own parsers.

Even though X3D offers many opportunities for 2D and 3D model development, we are studying Scalable Vector Graphics (SVG) as an additional mode of presentation for 2D graphics, possibly within X3D worlds. SVG will permit us to create more traditional 2D models, in addition to the 3D ones. Moving from 2D to 3D should ideally be a simple matter of changing stylesheets. Presentation is not limited to visual output. There are also opportunities to incorporate the addition of sound, music, and narrative to augment the visual experience.

Moreover, we have launched a new tangible user interface study to allow modelers the freedom of building dynamic models using wood and plastic blocks. We are planning on supporting both ultrasound and computer vision for determination of block identification, position, and orientation. The blocks will be positioned within a PVC “cage” and then sensed using the camera and sensors.

3.2. Model Specification Layer: MXL

MXL⁵ is an XML language for representing dynamic models. In many ways, MXL is the XML counterpart to the visual models created earlier using OOPM, except that MXL specifies no presentation information. A tenet of SGML and XML is that content and presentation should be separate. Adhering to this tenet allows modelers to choose custom presentation styles for their model components. An exploration of alternative aesthetics is one of the goals of *rube*, and so XML plays a central role.

XML schemas allow for XML-based models to be defined in terms of their formal grammars. MXL has a schema for Finite State Machines (FSM) and Functional Block Models (FBM). Therefore, more work is needed to populate this list with additional model types such as Petri nets and System Dynamics models. Also, we need to document ways in which stylesheets can be drafted to accomplish new model types. When we started work on MXL, we tried to parse it directly to code, but found this to be somewhat unwieldy based on the complexity inherent within multimodels. So, we developed a new, lower level language to support multimodel execution, DXL. MXL is translated into DXL using an XML stylesheet, and with the aid of the XSL transformation (XSLT).

There is a dashed line separating the first two blocks in Fig. 1. This line suggests that future work exists in more easily connecting the scene with the dynamic model. To date, this connection represents one of name correspondence, where for every object in the MXL model file, there must be a corresponding name in scene. If there are additional objects within the scene, that is to be expected since many scene objects represent components of the phenomenon being modeled, and not the model itself. Name correspondence is a weak approach to making *rube* work, but it is effective. It requires that everything be done manually. And, yet, most Computer Aided Design constructions are done manually. Can anything be done to semi-automate the task of scene construction based on the model? The XML stylesheet transformation may be ideally suited to such a task. Equivalently, what can be done to make it possible for the author to create dynamic models directly within a scene-based package? These represent two future areas where the author is relieved of having to create both files. Before, we embark upon this, we need to ensure a robust MXL implementation.

Another item that we require on top of MXL is an ontology-based representation of the more encompassing model relations. If one 3D object serves to represent the dynamics of another, we should ideally consider forming a semantic network to encode these semantics. We are reviewing the XML-based Resource Description Framework (RDF) as one method for constructing the network. Another approach is the DARPA sponsored Agent Markup Language (DAML).⁶

3.3. Model Assembly Layer: DXL

The Dynamics eXchange Language (DXL)⁷ encodes system dynamics using a simple block language. Each block either refines into a block network or it is terminated by a program (i.e., code), which can be either JavaScript or Java. It is here, in DXL, where XML is parsed into code. After some study with XSLT versus the XML Document Object Model (DOM), we settled on DOM as the method for parsing DXL into code. Work on DXL is still fairly new, and DXL has been demonstrated to take a series of fairly simple FBMs, created in MXL, and then to translate these into Java.

DXL has not been tested on, nor supports, multimodeling but this is our next goal. Also, we hope to support XML-oriented pipelines as part of DXL blocks where each block would be capable of operating upon XML input, and producing XML output. We envision that this may allow for more flexible simulation opportunities for better modeling the dynamics associated with changing geometry. XML would become a first class “data type” operated upon by DXL block networks.

3.4. Program

The result of using DOM on DXL presently yields a Java program. The DXL processor parses the DXL document, imports what is necessary from the next block (SimPackJ/S), and creates a program. The program is essentially a single event loop, which posts all DXL blocks on a future event list for sequencing, and then processes each one. A byproduct of processing a block is the scheduling of new events corresponding to outgoing links from that block. This method is similar to what was used in OOPM. The design of *rube* doesn’t prohibit the use of other target languages in the future, such as Python, Perl, or any other language supported by the programs that render 2D and 3D graphics, while supporting scripting for scene graph updating.

3.5. SimPackJ/S

SimPackJ/S⁸ represents the evolution of SimPack from the days of its C and C++ incarnations. The “J/S” suffix refers to a dual encoding in both Java and JavaScript. Java is most useful for speed and cross-platform compatibility. JavaScript, while slower than Java, is well-supported by Netscape and Internet Explorer as the default scripting language. Also, within existing VRML clients, JavaScript enjoys full support. DXL uses the event handling capability of SimPackJ/S when it outputs a program. Event scheduling turns out to be an efficient mechanism supporting multimodel execution. The primary work on SimPackJ/S is coming to a close, so that we can use it for DXL research. Since SimPackJ/S represents an “assembly layer” for *rube*, we have also performed work on taking MathML difference and differential equations and translating these to Java and JavaScript. This will yield a future capability for DXL block circuits to use these programs, generated by a SimPackJ/S “MathML component.”

3.6. Model Fusion Engine

The model fusion engine, through the use of XSLT, performs the task of combining the scene with the dynamics to produce a scene containing embedded dynamics. Thus, the fusion is one of combining geometry with dynamics. There are two ways in which X3D and VRML support dynamics: 1) Script node, and 2) External. When a script node is created, the VRML plugin processes the code and makes the appropriate real-time changes to the scene graph. To date, this has been our primary mode of achieving change for VRML worlds. The external approach can use VRML’s External Authoring Interface (EAI) or the more recent DOM API.

The *rube* architecture, and the fusion engine, are consistent with other approaches that use XML to encode knowledge about a domain. For fusing sensor information, it is necessary to fuse both data and knowledge about the sensors themselves. For example, SensorML,⁹ specifies a way to encode knowledge about sensors for earth observation.

3.7. 3D Scene with dynamics: X3D

The result of the previous fusion is a combined scene graph with code that makes changes to the scene graph and simulates the dynamic model in MXL. This is the termination of the *rube* process, as its main goal of fusing both dynamics and geometry within an immersive environment has been achieved.

4. RELATED WORK

The *rube* Project bears relationships with the research of others in the simulation field. *rube* differs mainly in its focus on XML and the concomitant strategy inherent within XML of separation of content from presentation, encouraging customized model forms such as those in 3D that are at the center of our work. Thus, the end result of our efforts, in terms of *customized model presentation*, is quite different from the goals of two other major research projects, Modelica¹⁰ and DEVS.¹¹ Let’s first consider Modelica. Modelica is an object-oriented language for modeling engineering systems. It has a strong “control engineering” flavor based on its lineage, and supports a low-level assembly “DSBlock” similar to a “Block” in DXL. Our plans are to go further with DXL by supporting XML as a first class “data” type, but as it stands today, DXL blocks bear strong resemblance to DSBlocks. Modelica tends to have broad industry support, especially in Europe, and tends to focus on providing especially thorough support for engineering systems requiring continuous-time numerical algorithms. Simulink¹² is a commercial system offering similar support.

Likewise, DEVS was originally cast as a formal system-theoretic specification for the semantics of discrete event systems, but has recently broadened into a standardization effort, as Modelica did earlier. DEVS, on the other hand, creates an equivalent project for *discrete event dynamic systems*, even though extensions exist for combined models and multimodels.

To date, these systems do not have an XML presence, however, we imagine that this will change as XML is viewed as a key language for model interchange. *rube* is constructed from XML, and with the express ideal of supporting customized model presentation. Therefore, *rube*’s purpose, when contrasted against Modelica and DEVS, lies closer to the human-model interface. *rube* embodies the separation of content from presentation, and especially encourages 3D presentation of dynamic model structure. Our complementary work in combining feature of Fine Art with Computing finds a home in the *rube* system, since students are able to construct their own 3D metaphors of familiar mathematical structures, including software.¹³

5. WEB SITES

- rube Project Page: <http://www.cise.ufl.edu/~fishwick/rube>
- VRML Links Page: <http://www.cise.ufl.edu/~fishwick/vrml>
- Digital Arts & Sciences Program: <http://www.digitalworlds.ufl.edu>
- Web3D: <http://www.web3d.org>

ACKNOWLEDGMENTS

Work on model fusion and next generation modeling is supported by the Air Force Research Laboratory under grant F30602-01-1-0592, and the National Science Foundation under grant EIA-0119532. Thanks to my current students, Taewoo Kim, Jinho Lee, and Minho Park for their ongoing work on *rube*, and to Prajakta Ugrankar for her architectural models.

REFERENCES

1. P. A. Fishwick, *Simulation Model Design and Execution: Building Digital Worlds*, Prentice Hall, 1995.
2. P. A. Fishwick, R. M. Cubert, and K. Lee, "A Multimodeling Framework for Distributed Simulation," tech. rep., University of Florida, August 1999. Air Force Research Laboratory AFRL-IF-RS-TR-1999-181.
3. P. A. Fishwick and J. Hopkins, "On Web-Based Models and Repositories," tech. rep., University of Florida, January 2001. Air Force Research Laboratory AFRL-IF-RS-TR-2001-3.
4. P. Fishwick, "Next Generation Modeling: A Grand Challenge," in *Proceedings of the 2002 Western Simulation Multiconference*, (San Antonio, TX), January 2002.
5. T. Kim and P. A. Fishwick, "An XML-Based 3D Model Visualization and Simulation Framework for Dynamic Models," in *Enabling Technologies for Simulation Science*, SPIE, (Orlando, FL), April 2002.
6. "Distributed Agent Markup Language." <http://www.daml.org>.
7. J. Lee and P. A. Fishwick, "A Dynamics Exchange Language Layer for RUBE," in *Enabling Technologies for Simulation Science*, SPIE, (Orlando, FL), April 2002.
8. M. Park and P. A. Fishwick, "SimPackJ/S: A Web-Oriented Toolkit for Discrete Event Simulation," in *Enabling Technologies for Simulation Science*, SPIE, (Orlando, FL), April 2002.
9. M. Botts, "An XML-based Sensor Model Language (SensorML) for Earth Observing Dynamic Sensors," tech. rep., University of Alabama, March 2002. <http://vast.uah.edu/SensorML/>.
10. M. Tiller, *Introduction to Physical Modeling with Modelica*, Kluwer Academic, 2001.
11. B. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation*, Academic Press, 2nd ed., 2000.
12. "Simulink." <http://www.mathworks.com>.
13. P. Fishwick, "Aesthetic Programming: Making Artistic Software," *Leonardo*, 2002. to be published.