*Proceedings of the 2003 Winter Simulation Conference*
*S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.*

# RUBE: A CUSTOMIZED 2D AND 3D MODELING FRAMEWORK FOR SIMULATION

Paul Fishwick
Jinho Lee
Minho Park
Hyunju Shim

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

## ABSTRACT

We present a system called RUBE, which allows a modeler to customize model components and model structure in 2D and 3D. RUBE employs open source tools to assist in model authoring, allowing the user to visualize models with different metaphors. For example, it is possible to visualize an event graph as a city block, or a Petri network as an organically-oriented 3D machine. We suggest that such flexibility in visualization will allow existing model types to take on forms that may be more recognizable to modeling subcommunities, while employing notation as afforded by inexpensive graphical hardware. There is also a possibility to create model types using entirely new notations.

## 1 INTRODUCTION

The discipline of computer simulation involves an integrated set of components that are revisited, with feedback, and executed as required for analyzing real world systems. Generally, a simulation analyst proceeds by gathering data, formulating a hypothetical model for the system, turning this system into code, executing the code, gathering the resulting data, and performing statistical analyses. Feedback can occur anywhere, so that after one or several computer runs are made, this suggests changes to the model. Most models assume an a priori structure of some sort: linear system, discrete event model type, differential equations. One of the tasks of simulation is model generation, and this involves a choice of model type and notation. Our purpose in this paper is to describe a system that we call RUBE (Fishwick 2002), which allows significant freedom in customizing notation for models, and then executing them while leveraging the environment governed by the notation. So, for example, one might choose to define an event graph or a Petri net but then display this model using 3D components, and have the simulation execution feedback some of its results to animate the model as appropriate. The reason for this type of customization is to allow modelers greater flexibility and freedom in representation, so that the metaphors that underlie all models can be brought forth and surfaced as required.

RUBE is constructed as a kind of process, with process components. At first a model is constructed in 2D or 3D. For all models, we have subscribed to the idea of *open source* modeling and simulation environments, which aids in dissemination of the modeling software. For 2D, we are using *SodiPodi* and for 3D, *Blender*. In a pure modeling building environment, model components could be chosen from a library with full prior knowledge of their semantic functions. While our research is moving us in that direction, currently we must provide an environment that manually associates a 2D or 3D icon with its semantic meaning. So, a sphere might be a state or event, but the tool doesn't know this ahead of time, so the modeler must specify it during the model authoring procedure. Additionally, the modeler must specify the semantic function associated with a model component. When in a state of a finite state machine, for example, the modeler will surely want code to be executed during that state's duration. This code must be associated with the state. Models are translated into an XML (eXtensible Markup Language) language that we call MXL (Multmodeling eXchange Language) (Kim and Fishwick 2002).

## 2 OVERVIEW

### 2.1 Related Work

There are visualization tools, which support the areas in Information Visualization, Software Visualization, and Model Visualization, and modeling tools for modeling and executing a simulation of complex systems.

Kirner and Martins developed the InfoVis system, which combined the Information Visualization and the Virtual Reality techniques to generate graphical representa-

tions of the information. A notable feature of the InfoVis was that it provided the characteristics of Virtual Reality: Interaction and Navigation. However, they employed different commercial software to support these characteristics (Kirner et al. 2000).

Another visualization tool used in the Software Visualization is GAMMATELLA, developed by Orso and colleagues. They defined a distinct visualization approach that provided a multilevel representation of a software system and a use of coloring to represent its data. The users could efficiently visualize and explore the large amount of data along with big programs using these techniques (Orso et al. 2003).

RUBE, as a Model Visualization tool, focuses on the customization of a model structure. In RUBE, the users are allowed to customize their own structures, such as adding sounds, textures and objects, in appropriate areas. The customization needs an explicit support in the way of XML languages, and edits to open source packages and it leads ultimately to an *integrative modeling interface* capability where different models can be laid on top of, or morphed-to, the base scene being modeled.

Modelica is an object-oriented modeling language for complex and heterogeneous physical systems. It supports hierarchical modeling and has the ability to represent the 2D-based topology in a high-level abstraction as well as detailed model execution by using equations for modeling the physical system (Mattsson et al. 1998; Elmqvist et al. 1999; Tiller 2001).

PtolemyII provides heterogeneous, concurrent modeling and design, and it is based on MoML to specify each model. MoML is an XML-based functional block modeling language and supports multimodeling for PtolemyII (Buck et al. 1994; Lee et al. 2000; Liu et al. 2002).

Modelica is a 2D-based toolkit and is not an XML-based language. It does not provide any customization for the model visualization nor does it directly support reusing or exchanging of models. PtolemyII supports 2D and partially, 3D (Java 3D) visualization, whereas RUBE fully supports XML-based 2D(SVG) and 3D(X3D) web-based visualization and simulation (Fishwick 1996; Page et al. 1998; Page et al. 2000) as well as a multimodel (Fishwick and Zeigler 1992; Nance et al. 1999; Zeigler et al. 2001). The block diagram of MoML is composed of general functional blocks that are more complex than the simple blocks of DXL in RUBE (Lee and Fishwick 2002).

### 2.2 RUBE Framework

A simplified overall structure of the XML-based RUBE framework is shown in Figure 1. RUBE is an XML-based framework and application, which permits the users to specify and simulate a dynamic model, with an ability to customize a model presentation using 2D or 3D visualization. In Figure 1, the RUBE framework is defined using two stages: model representation and model creation.

For model representation, a dynamic model, which is generated by a 2D or a 3D interface, is composed of two sorts of files: a scene file, which contains 2D or 3D geometry objects, and a model file, which is represented by MXL. RUBE uses *Sodipodi* and *Blender* for each representation of the 2D or the 3D model scenes and behaviors. The scene files don't have any information about model behavior or dynamics except regarding the appearance of the model. The scene files can be either 2D or 3D XML documents: SVG (Scalar Vector Graphics) or X3D (eXtensible 3D) respectively. The MXL file describes the behavior of the model to represent the model file that describes a heterogeneous multimodel in an abstract level such as FBM (Functional Block Model) and FSM (Finite State Machine).

For model creation, a 2D or a 3D merge engine, which uses XSLT (eXtensible Stylesheet Language Transformation), merges two XML documents: a scene file and a model file. For model simulation, the *MXLtoDXL* translator translates a model file written in MXL into an assembly level modeling language called DXL, which can be described with homogeneous simple block diagrams. The DXL is translated into an executable programming code for the model simulation using the *DXLtoJavascript* translator. The programming code can be executed using SimpackJ/S (Park

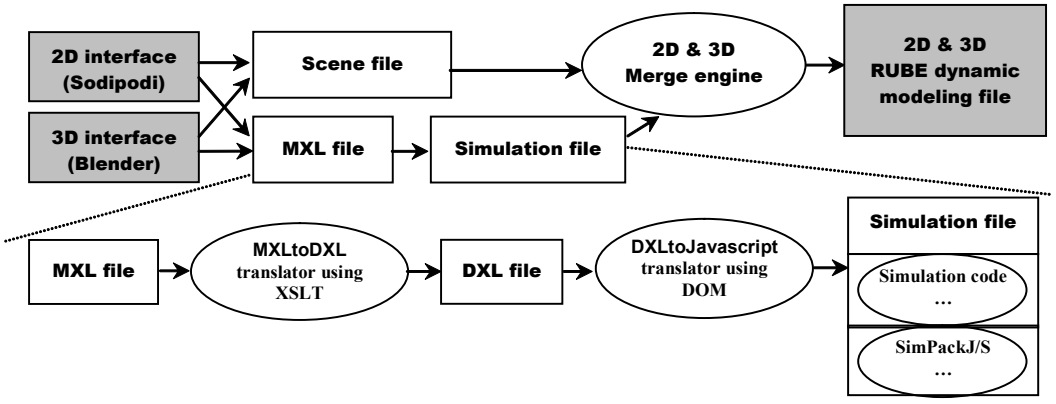



Figure 1: RUBE Framework

and Fishwick 2002), which provides the underlying code foundation libraries, classes, and objects.

## 3 RUBE METHODOLOGY

### 3.1 Example: A Four-Stroke Gasoline Engine

In this section, we will introduce a scene, a four stroke gasoline engine as an example of how to use RUBE. A four-stroke gasoline engine has four phases, which cycle until the engine is turned off. The four phases are *Compression*, *Ignition*, *Expansion*, and *Exhaustion*. In this gasoline engine, injected fuel vapor is compressed by the piston of the cylinder (*Compression*) and then the fuel is ignited (*Ignition*). As a result of the ignition, the piston of the cylinder moves back (*Expansion*). The resulting fumes exit through the exhaust manifold (*Exhaustion*).

The behavior of the engine can be described as the FSM in Figure 2. While an input value of 0 will let the engine stay at the current state, the value of 1 causes the engine to move to the next state. An input value of 2 in the *Ignition* state indicates that the ignition is turned off (*Off* state).

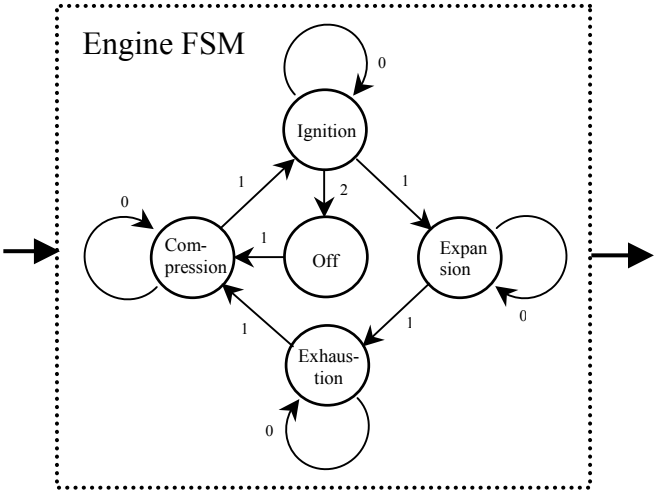



Figure 2: An FSM Describing a Four-Stroke Gasoline Engine

Figure 3 shows an SVG visualization of engine snapshots for each phase. The intake, compression, power or work, and exhaust stroke are mapped to an *Expansion*, *Compression*, *Ignition*, and *Exhaustion* state respectively. The position of the piston changes as the state of the engine changes.

The engine system can also be modeled with a pipeline of 3 functions to create an FBM. To execute the engine, input to and output from the engine must be specified from the outside world. The engine can be embedded inside of the FBM block with one input and one output block associated with it. The input block feeds the input value to the engine. The engine block represents an engine model

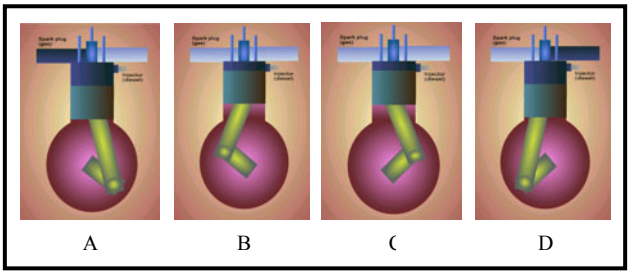



Figure 3: Phases of a Four-Stroke Engine in SVG A) Intake Stroke; B) Compression Stroke; C) Power or Work Stroke; D) Exhaust Stroke

that behaves as described in Figure 2. The current state of the engine is passed to the next block, namely the output block. Visualization of the engine in 2D and 3D will be discussed in section 3.2 and 3.3, respectively.

### 3.2 2D Model

2D model visualization in RUBE is achieved using SVG (Scalable Vector Graphics). SVG represents a 2D graphic information in a compact and portable form. It is a vector and text based 2D graphic language. Unlike raster graphics, SVG is in a vector format, which means SVG images can be printed at any resolution without loss of image quality, when one enlarges or reduces the size of the images.

As with other XML applications, an SVG file can be created from a text editor. The three types of graphic objects in SVG are vector graphic shape, image, and text. General SVG path elements and basic shapes enable the construction of both simple and complex 2D models. The basic shapes in SVG are the rectangle, circle, ellipse, line, plotlines, and polygon. The filtering operation gives a raster effect on the SVG drawings, while the graphics are still scalable and displayable at different resolutions. Once the developers understand the SVG elements and syntax, they are free to build their own personalized model and component representation.

SVG is relatively easy to create manually using a text editor; however, having a visual editor which creates SVG automatically is more useful. *Sodipodi* is the recommended SVG editor in RUBE since its native file format is SVG. *Sodipodi* provides a manual editing window from where users can manipulate SVG objects in detail, such as giving the object a meaningful ID. In addition, *Sodipodi* supports the Linux system, as well as the Windows system. Figure 4 shows a snapshot of *Sodipodi*.

SVG viewers include an XML parser, a CSS parser, and an SVG rendering engine to draw the images. Figure 5 shows rendered SVG of an FBM block diagram for a four stroke gasoline engine. Three different colored rectangles represent each block and two arrows, which consist of three path objects, represent the data flow.

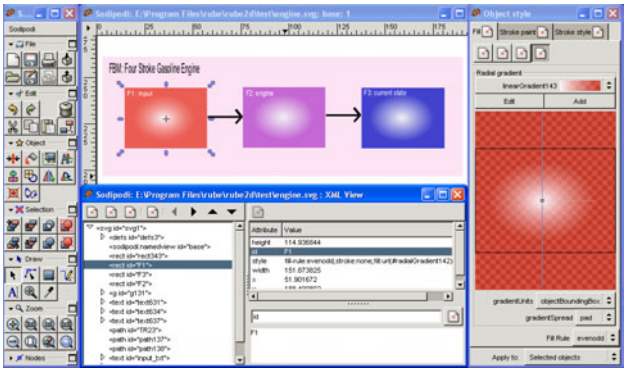


Figure 4: A Snapshot of the Sodipodi Interface

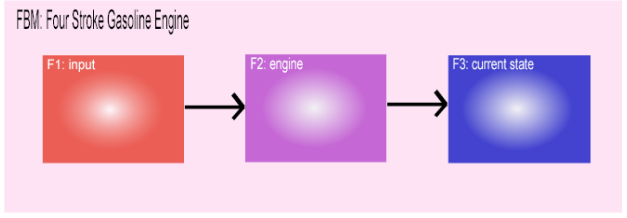



Figure 5: FBM for the Engine in SVG

Although SVG can be dynamic and interactive, the SVG scene discussed in this section is static, without the dynamic behavior associated with it. This SVG only visualizes the engine system. Adding a dynamic behavior to SVG drawings can be done by writing a script file.

The JavaScript code from SimPack J/S and the simulation code generated based on the model file are integrated with the scene file. As a result of a dynamic behavior, the code is added to the static scene file. Figure 6 shows the simulation output of the engine model from the 2D authoring process. Text in the input block and the current state block represent values of each block at simulation time 99. The trajectory of input to the engine block is drawn in the engine input graph, and the trace of the current state over time is shown in the engine state graph.

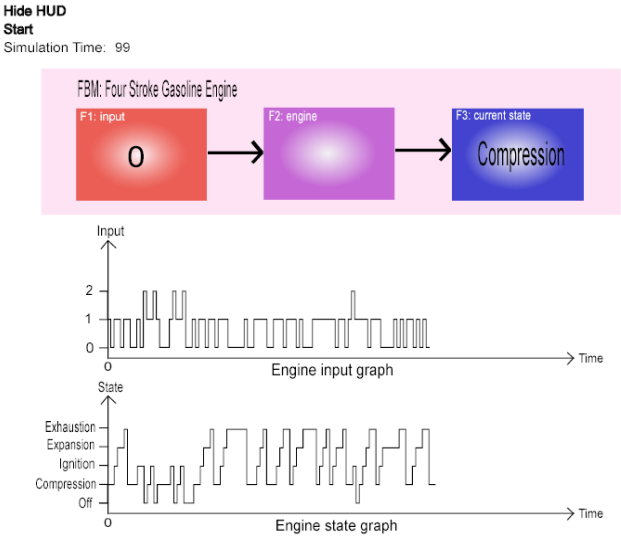



Figure 6: Simulation Output from the 2D Engine Model

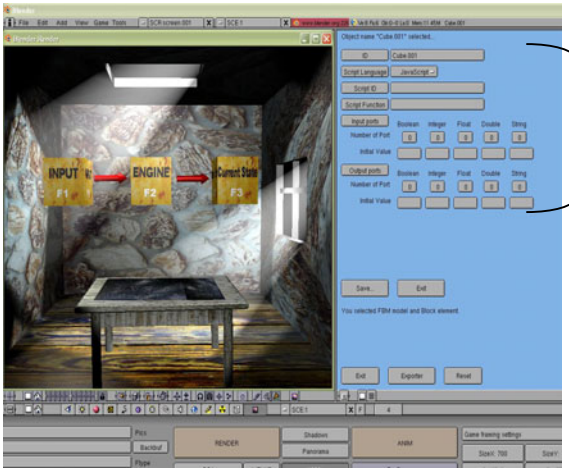
### 3.3 3D Model

We use Blender as a 3D authoring tool and the Python scripting language for implementing an interface.

Blender is a free and a fully functional 3D modeling/rendering/animation package for Linux/Unix/Windows systems. Python is an interpreted, interactive, and object-oriented programming language that is often compared to Tcl, Perl, Scheme or Java. Python combines remarkable power with very clear syntax, which is also usable, as an extension language, for applications that need a programmable interface. Python is included in Blender as a scripting language, which allows us to use Python script to export a scene graph into different files, such as MXL (Multimodel eXchange Language) and VRML (Virtual Reality Modeling Language).

We have developed an interface to author a Blender 3D scene within the RUBE framework. The Blender interface for RUBE consists of 2 parts: a *Scene recognizer* and an *Exporter*. The *Scene recognizer* is a process that assists the user in mapping each object of a Blender scene to MXL elements and attributes. This provides a *gateway* for converting 3D objects into text forms, called MXL. A snapshot of the Scene recognizer is shown in Figure 7. For example, as a user selects an object in a scene and moves a mouse to an interface window in Blender, an attribute editor for the object appears dynamically. The user then chooses a model type, such as FBM, FSM, and Petri-Net, and an element type, such as block, state, trace, and transition. Currently FBM and FSM are supported as model types. In addition, we provide a simulation element type that is essential for model execution within an experimental frame. The contents of the attribute editor are changed when a user presses the "continue" button. In Figure 7, if a user selects an arrow and presses the button, the contents of the attribute editor is changed. The new attribute editor requests more specific data from the user, such as the source object and the destination object. In this case, the user can designate an object as the source or the destination in the scene by (1) choosing an object, which may be a source or a destination object, (2) moving the mouse to the interface window, and then (3) pressing a source or a destination button provided by the editor. All data given by the user is stored in the dictionary data structure provided by Python. This data can then be retrieved and modified when the user selects the same object again.

*Exporter* is a collection of components that can create a final dynamic scene file, VRML or Blender, using the user's Blender scene file, given input data and the user-defined script files in JavaScript or Python source files. Currently only the VRML scene is generated as a final dynamic scene. Figure 8 shows the final VRML scene created by the interface.




***Object attribute editor***

**ID**: Each object name
**Script ID**: User-defined Script file name
**Function**: function name in the script file
**Input port**: Input type and a number of ports
**Output port**: Output type and a number of ports
**Source and Destination**: Object's source and destination

Figure 7: Scene Recognizer

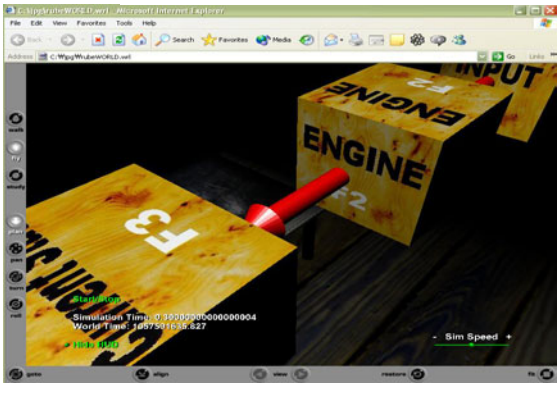



Figure 8: Final VRML Scene Generated by the 3D-Based RUBE Framework

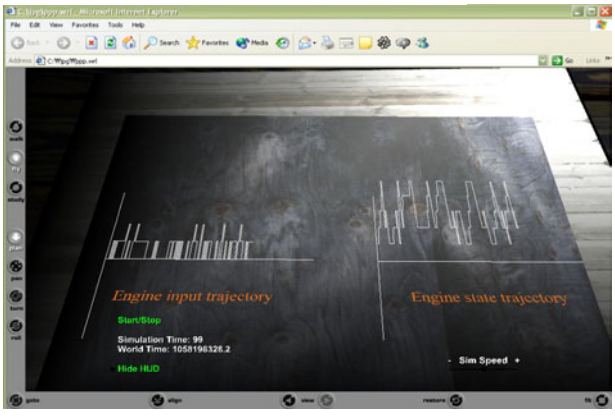



Figure 9: Simulation Output from the 3D Engine Model

In the 3D-based RUBE framework, we provide two kinds of simulation methodologies: a real-time method using JavaScript in VRML, and a non-real-time method using Python in Blender. In the real-time simulation, the final dynamic scene file is VRML with simulation codes in JavaScript. However, in the non-real-time simulation, the final dynamic scene file is Blender with simulation codes in Python. In the VRML case, VRML has an ability to interact with the user. The user can start, stop, or resume model simulation by touching the 3D objects. Then VRML allows the user to navigate the virtual world. In contrast, Blender provides higher quality rendering, and various kinds of 3D object types such as sub-division polygon meshes and NURBS surfaces.

Figure 9 shows the simulation output of the engine model under the real-time simulation environment. Two graphs in Figure 9 represent an input trajectory and the corresponding state trajectory.

### 3.4 XML-based Modeling and Simulation

MXL and DXL are XML-based modeling languages used for representing and executing models in the RUBE framwork. MXL is an XML-based modeling language defined for representing the semantics and the behaviors of a heterogeneous dynamic model such as a multimodel as well as a simple FSM and FBM. DXL is a low-level XML-based modeling language in RUBE. It represents a homogeneous dynamic model using a simple block diagram, which consists of *blocks*, *ports*, and *connectors*. Because both MXL and DXL are XML-based, MXL models can be easily translated into DXL models through XSLT-based translators.

Figure 10 shows an MXL file for a four stroke gasoline engine which is used for a 2D and a 3D modeling and simulation. In Figure 10, <*input*> and <*output*> handles are used to connect different blocks and to transfer data between blocks in Figure 11. Because these blocks can include another model for a multimodeling, these handles play a role of connecting heterogeneous models. The element <simulation> provides the factors for simulating a model such as starting time, ending time, and time duration.

```
<?xml version="1.0" encoding="utf-16"?>
<MXL>
 <fbm id="MXL">
  <block id="F1">
   <output id="F1_outports_integer1" … index="0"/>
   <script lang="JavaScript" src="input.js" func="gen"/>
  </block>

  <block id="F2">
   <input id="F2_inports_integer1" … index="0"/>
   <output id="F2_outports_integer1" … index="0"/>
   <script lang="JavaScript" src="input.js" func="engine"/>
  </block>

  <block id="F3">
   <input id="F3_inports_integer1" … index="0"/>
   <script lang="JavaScript" src="input.js" func="print"/>
  </block>
…
 </fbm>
 <simulation start_time="0" end_time="100"
              delta_time="1" cycle_time="1" />
</MXL>
```

Figure 10: MXL Specifying the Engine's Dynamics

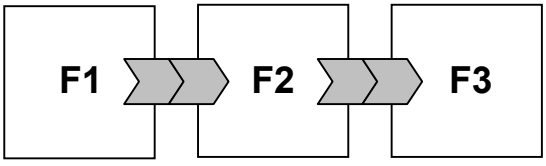



Figure 11: FBM Model with Block Handles

DXL is based on functions that are translated into basic code units of SimPackJ/S using event-based scheduling (Fishwick 1995; Nance 1996; Zeigler 1984). Functional blocks of DXL support heterogeneous models with *synchronous* and *asynchronous* properties. *Asynchronous* blocks can process the input data, when any of their input ports has valid data, or send output to next blocks, when any of their output ports has valid data. On the other hand, *synchronous* blocks can only process the input data when *all* of their input ports have valid data or can only send output to next blocks when *all* of their output ports have valid data.

The simple blocks of DXL are actual program codes such as Java or Javascript. They are executed by an event-based scheduling method, which regards each simple block as a unit of the simulation. DXL operates like an assembly language layer for RUBE, by providing independence between various heterogeneous model types and their actual simulation codes.

## 4 CONCLUSION

In this paper, we presented a new modeling and simulation framework called RUBE. We explained 2D and 3D RUBE methodologies with a four-stroke gasoline engine model. The primary goal of RUBE is to provide the extensive modeling and the simulation environment for modelers, to achieve personalized, customized, and aesthetic modeling in a 2D and a 3D model. RUBE also provides a convenient and modularized Web-based modeling and simulation using emerging XML technologies.

While the above advantages of RUBE have given us a novel modeling environment, allowing simulation models to break out of minimal, standard iconography, we are currently working on a number of improvements to counter the deficits of our approach. The primary disadvantage of RUBE is that it is a little cumbersome to build models compared with a special-purpose model building tool. The price of customization and flexibility seems to be slower speed in model creation. There are specific techniques that we are researching to assist in solving this problem. The first involves a set of libraries for both model components and functions. Let's take a finite state machine as the model type under construction for an example. All states and transitions should come with default functions and geometry. If the user wishes to make modifications, this is made easier when defaults are present. So, the 3D defaults might be spheres and 3D arrows. Default functions might be to plot the current state over time as the machine output. If the user chooses a special sort of geometry to represent a state (such as a particular geometrical object) then it should be possible to reuse that object in the model without explicitly specifying that it is also a state. The assumption here is that for many classes of models, similar model component types suggest similar geometry. Another problem is the issue of topological connectivity, which must be manually specified in *SodiPodi* and *Blender* since they know nothing of connectivity. A solution is to induce the topology from the geometry using an intelligent algorithm.

Recently, we have been interested in a new modeling environment for future work. We refer to this environment as the *integrative modeling interface*, which allows the user to switch between information, geometry, and dynamic models while never leaving the immersive scene. This will require more advanced visualization techniques and involve a more complicated multimodel environment. In addition, we will support the processing of an XML data stream in a future DXL and a distributed simulation based on a message passing method. In other words, each block can operate on an XML input stream and produce an XML output. These blocks can become the processes in a distributed simulation system. This means a model itself can be processed as a parameter of blocks in DXL so that multiple models can communicate their data using message passing.

## ACKNOWLEDGMENTS

We would like to thank the National Science Foundation under grant EIA-0119532 and the Air Force Research Laboratory under grant F30602-01-1-05920119532 for support of this research.

## REFERENCES

J. T. Buck, Ha S., E. A. Lee, and D. G. Messerschmitt. 1994. Ptolemy: A Framework for Simulating and Pro-

totyping Heterogeneous Systems, *Int. Journal of Computer Simulation, special issue on Simulation Software Development*, 4:155-182.

H. Elmqvist, S.E. Mattsson, and M. Otter. 1999. Modelica - A Language for Physical System Modeling, Visualization and Interaction, Plenary paper. *IEEE Symposium on Computer-Aided Control System Design*, CACSD'99, 630-639. Hawaii.

P. A. Fishwick and B. P. Zeigler. 1992. A Multimodel Methodology for Qualitative Model Engineering, *ACM Transactions on Modeling and Computer Simulation*, 2(1):52-81.

P. A. Fishwick. 1995. *Simulation Model Design and Execution: Building Digital Worlds*. Englewood Cliffs, NJ: Prentice-Hall.

P. A. Fishwick. 1996. Web-Based Simulation: Some Personal Observations, *Proceedings of the 1996 Winter Simulation Conference*, 772-779. San Diego, CA.

P. A. Fishwick. 2002. *rube*: An XML-Based Architecture for 3D Process Modeling and Model Fusion, *Proceedings of Enabling Technology for Simulation Science*, Part of SPIE Aerosense '02 Conference, 330-335. Orlando, FL.

T. Kim and P. A. Fishwick. 2002. An XML-Based Visualization and Simulation Framework for Dynamic Models. *Proceedings of Enabling Technology for Simulation Science*, Part of SPIE Aerosense '02 Conference, 336-347. Orlando, FL.

T. G. Kirner and V. F. Martins. 2000. Development of an Information Visualization Tool Using Virtual Reality. *Proceedings of the 2000 ACM symposium on Applied computing,* 604-606.

E. A. Lee and S. Neuendorffer. 2000. MoML - A Modeling Markup Language in XML, Version 0.4, *Technical Memorandum UCB/ERL M00/12*, University of California, Berkeley, CA 94720.

J. Lee and P. A. Fishwick. 2002. A Dynamic Exchange Language Layer for *rube*, *Proceedings of Enabling Technology for Simulation Science*, Part of SPIE Aerosense '02 Conference, 359-366. Orlando, FL.

X. Liu, J. Liu, J. Eker, and E. Lee. 2002. Heterogeneous Modeling and Design of Control Systems, to appear in *Software-Enabled Control: Information Technology for Dynamic Systems*, New York, IEEE Press.

S. E. Mattsson, H. Elmqvist, and M. Otter. 1998. Physical System Modeling with Modelica, *Control Engineering Practice* 6:501-510,.

R. E. Nance 1996. A History of Discrete Event Simulation Programming Languages, In *History of Programming Languages*, ACM Press and Addison-Wesley Publishing Co., 369-427.

R. E. Nance, E. M. Overstreet, and E. H. Page. 1999. Redundancy in Model Specification for Discrete Event Simulation. *ACM Transactions on Modeling and Computer Simulation* 9(3):254-281.

A. Orso, J. Jones, and M. J. Harrold. 2003. Visualization of Program-Execution Eata for Deployed Software. *Proceedings of the 2003 ACM symposium on Software visualization*, 67-76.

E. H. Page, P. A. Fishwick, K. J. Healy, R. E. Nance, and R. J. Paul. 1998. The Modeling Methodological Impacts of Web-Based Simulation. *Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation, Simulation Series*, Volume 30, Number 1, P. A. Fishwick, D. R. C. Hill, and R. Smith, Eds. Society for Computer Simulation, 123-130. San Diego, CA.

E. H. Page, A. Buss, P. A. Fishwick, K. J. Healy, R. E. Nance, and R. J. Paul. 2000. Web-Based Simulations: Revolution or Evolution? *ACM Transactions on Modeling and Computer Simulation*, 10(1):3-17.

M. Park and P. A. Fishwick. 2002. SimPackJ/S: A Web-Oriented Toolkit for Discrete Event Simulation, *Proceedings of Enabling Technology for Simulation Science*, Part of SPIE Aerosense '02 Conference, 348-358. Orlando, FL.

M. Tiller. 2001. Introduction to Physical System Modeling with Modelica. *Kluwer International Series in Engineering and Computer Science*, 615, Kluwer Publishing.

B. P. Zeigler. 1984. *Multifacetted Modeling and Discrete Event Simulation*, Academic Press.

B. P. Zeigler, H. Praehofer, and T. G. Kim. 2001. *Theory of Modleing and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Second Edition, Academic Press.

## AUTHOR BIOGRAPHIES

**PAUL FISHWICK** is Professor of Computer and Information Science and Engineering at the University of Florida. He received the PhD in Computer and Information Science from the University of Pennsylvania in 1986, and has six years of industrial and government production and research experience (Newport News Shipbuilding and NASA Langley Research Center). His research interests are in computer simulation modeling and analysis methods for complex systems. He is a Senior Member of the IEEE and a Fellow of the Society for Computer Simulation. He is also a member of the IEEE Society for Systems, Man and Cybernetics, ACM and AAAI. Dr. Fishwick founded the comp.simulation Internet news group (Simulation Digest) in 1987, which has served numerous subscribers. He has chaired several workshops and conferences in the area of computer simulation, including serving 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 delivered 10 Keynote addresses at major conferences relating to simulation. He has published over 125 technical publications, written one textbook, co-edited two Springer Verlag volumes in simulation, and published seven book chapters. His email and web addresses are <fishwick@cise.ufl.edu> and <http://www.cise.ufl.edu/~fishwick>.

**JINHO LEE** is a Ph.D. student in Computer and Information Science and Engineering at the University of Florida. He received his M.S. in Computer and Science Engineering from Sungkyunkwan Univerity in 1999. His research interests are modeling methodology and distributed system. His email address is <jhlee@cise.ufl.edu>.

**MINHO PARK** is a PhD student of Computer and Information Science and Engineering at the University of Florida. He received his M.S. in Computer and Information Science and Engineering at the University of Florida in 2002. He worked as a software engineer at Korea Securities Computer Corporation in Korea, from 1994 to 1998 and an assistant manager at Good-Morning Securities Company in Korea, from 1999 to 2000. His major research areas are modeling for computer simulation and 3D Model Visualization and customization. His email address is <mhpark@cise.ufl.edu>.

**HYUNJU SHIM** is a Ph.D. student in Computer and Information Science and Engineering at the University of Florida. She received her M.S. in Computer and Information Science and Engineering from the University of Florida in 2003. Her research interests include 2D Model Visualization, customization, and simulation. Her email address is <hshim@cise.ufl.edu>.