VISUAL AGENT-BASED MODEL DEVELOPMENT WITH REPAST SIMPHONY

M.J. NORTH,* Argonne National Laboratory, Argonne, IL,
and The University of Chicago, Chicago, IL
ERIC TATARA, Argonne National Laboratory, Argonne, IL
N.T. COLLIER, Argonne National Laboratory, Argonne, IL,
and PantaRei Corp., Cambridge, MA
J. OZIK, Argonne National Laboratory, Argonne, IL

ABSTRACT

Repast is a widely used, free, and open-source agent-based modeling and simulation toolkit. Three Repast platforms are currently available, each of which has the same core features but a different environment for these features. Repast Simphony (Repast S) extends the Repast portfolio by offering a new approach to simulation development and execution. This paper presents a model of physical infrastructure network interdependency as an introductory tutorial and illustration of the visual modeling capabilities of Repast S.

Keywords: Agent-based modeling and simulation, Repast, toolkits, and development environments

INTRODUCTION

Repast (ROAD 2005, North, Collier, and Vos 2006) is a widely used, free, and open source agent-based modeling and simulation toolkit with three released platforms, namely Repast for Java, Repast for the Microsoft .NET framework, and Repast for Python Scripting. Repast Simphony (Repast S) extends the Repast portfolio by offering a new approach to simulation development and execution, including a set of advanced computing technologies for applications such as social simulation. North et al. (2005a and 2005b), Howe et al. (2006), and Parker et al. (2006) provide an overview of the Repast S runtime and development environments.

We use a model of networked physical infrastructure to demonstrate the visual design capabilities of the Repast S toolkit and as an introductory tutorial. While the example is not intended to model real phenomena, the model's complexity is high enough to illustrate how the user may develop multi-agent models.

It is important to note that Repast S and its related tools are still under development. This paper presents the most current information at the time it was written. However, changes may occur before the planned final release.

* *Corresponding author address*: Michael J. North, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439; email: north@anl.gov.

THE REPAST S MODEL IMPLEMENTATION BUSINESS PROCESS

As discussed in North et al. (2005a and 2005b), the Repast S model implementation business process is as follows:

- The modeler creates model pieces, as needed, in the form of plain old Java objects (POJOs), often using automated tools or scripting languages such as Groovy.
- The modeler uses declarative configuration settings to pass the model pieces and legacy software connections to the Repast S runtime system.
- The modeler uses the Repast S runtime system to declaratively tell Repast S how to instantiate and connect model components.
- Repast S automatically manages the model pieces based on (1) interactive user input and (2) declarative or imperative requests from the components themselves.

The POJO model components can represent anything but are most commonly used to represent the agents in the model. While the POJOs can be created by using any method, this paper discusses one powerful way to create POJOs for Repast S: the Repast S development environment. However, modelers can use any method—from hand coding to wrapping binary legacy models to connecting into enterprise information systems—to create the Repast S POJO model components.

Regardless of the source of the POJOs, the Repast S runtime system is used to configure and execute Repast S models. North et al. (2005b) detail the Repast S runtime system, the design of which includes:

- Point-and-click model configuration and operation;
- Integrated two-dimensional, three-dimensional, and other views;
- Automated connections to enterprise data sources; and
- Automated connections to powerful external programs for conducting statistical analysis and visualizing model results.

SIMPLE PHYSICAL INFRASTRUCTURE NETWORK MODEL

A model of interconnected physical infrastructure networks is presented as an introductory tutorial and illustration of the visual modeling capabilities of Repast S. The model consists of a natural gas transmission and DC electric power network (Tatara et al. 2007b). The natural gas transmission model consists of a network of interconnected links and nodes, where the nodes function as delivery, receipt, and/or pipeline termination points, and the links function as gas pipelines that transport natural gas between nodes. The DC electric network model considers a balance of demand and generation given the transmission topology. The nodes in the

electric network represent generators and load points, while the links function as electrical transmission lines. The two networks are connected via links between the natural gas network and gas-fired electric power plants (generators) in the electrical network. The simple networks presented here will model propagation of pressure and power along the gas and electrical networks.

VISUAL PROJECT CREATION AND AGENT DESIGN

While previous versions of Repast required the user to set up and configure an appropriate integrated development environment (IDE), Repast S provides a preconfigured Eclipse-based IDE such that no *a priori* programming experience is required to build a model. Although the Repast IDE is aimed at novice developers, the full Repast S Java application programming interface (API) and advanced IDE configuration options are available at any time. Previously, Tatara et al. (2006) discussed using Repast S to create a model of wolf-sheep predation through the Java API.

After the Repast S IDE is started, the user may choose to continue working with an existing project or create a new project. The project creation wizard prompts the user for the type of project to create (Figure 1). The user is prompted for basic project information such as the project name (Figure 2). Additional project options are available to the advanced user, although these options may simply be left as the default.

When the Repast project is created in the workspace, a set of project components is visible in the package explorer, shown on the left side of Figure 3. These components include things such as directories for storing user data and the project source code. Also visible in Figure 3 is the Score editor, which specifies the hierarchical structure of the model contexts, agents, and projections. Model elements are represented graphically in a tree, and components may be added on a point-and-click basis. Once model elements have been placed in the Score editor, their properties may be edited in the Properties window shown at the bottom of Figure 3.

After the model Score has been completed, the user may start creating the agent objects. At this point, the advanced user may chose to create the agent classes using the Java API, while those users not familiar with Java may chose to use the Repast agent editor. The agent creation wizard is accessed via the package explorer and allows the user to create a number of Repast objects (Figure 4). When a new agent is created (Figure 5), the IDE view switches to the agent editor view as shown in Figure 6.

As discussed in detail in Ozik et al. (2007), the visual agent behavior editor, the new project wizard and the new agent wizard are modified forms of Alexander Greif's free and open source Flow4J-Eclipse components (Greif 2006) that have been adapted specifically for agent-based modeling. Greif (2006) has made the Flow4J-Eclipse system available under a BSD-style free and open source license. The Repast project team has built on Greif's contribution to create the above-mentioned Repast S components. From the Flow4J home page, Greif (2006) states:

> Flow4J is an Eclipse Plug-in for modeling process flows in a drag and drop manner. A process flow can contain process steps (I call them flowlets), which can be linked together [in]to a complex flow.

Ozik et al. (2007) provides the details on what Repast has both inherited and modified/adapted from Greif's (2006) Flow4J-Eclipse system.

Like the Flow4J-Eclipse visual editor, the Repast agent editor itself consists of an editable icon panel and a palette of behavior icons, which may be dragged into the edit panel and modified. As discussed in Ozik et al. (2007), the icons are analogous to blocks in a flowchart that may be connected in flexible ways to create the agent behavior logic. Figure 6 shows the creation of an agent property "pressure" for the GasNode agent class. The property parameters may be edited in the bottommost panel in Figure 6. The user is asked to specify a number of required elements such as the property name, data type, and initial values, while several optional data, such as a long description of the property, may also be defined.

Behaviors are defined by creating a behavior element in the workspace as shown in Figure 7. The behavior element may be either a scheduled behavior or one that is event-driven. The desired behavior for the nodes in the natural gas network is to react to changes in pressure upstream. Therefore, the behavior at the gas nodes will be event-driven and caused by changes in pressure from connected gas nodes.

The behavior block defines how and when the behavior occurs and not what actually happens next. A Task block is used to define the active part of the behavior to which it is associated (Figure 8). The Task block specifies what the agent does when the behavior is triggered. Continuing with the gas network node behavior, the task should adjust the node's pressure based on the pressure of the upstream node. Finally, the behavior logic is terminated with an End block as shown in Figure 8.

When the user saves the diagram, the Repast IDE automatically compiles the diagram into usable code that may be immediately loaded into the Repast runtime without the user ever needing to write Java code. The agent behavior editing step is repeated in this example (not shown) for the electric network nodes, using power rather than pressure as the propagated variable.

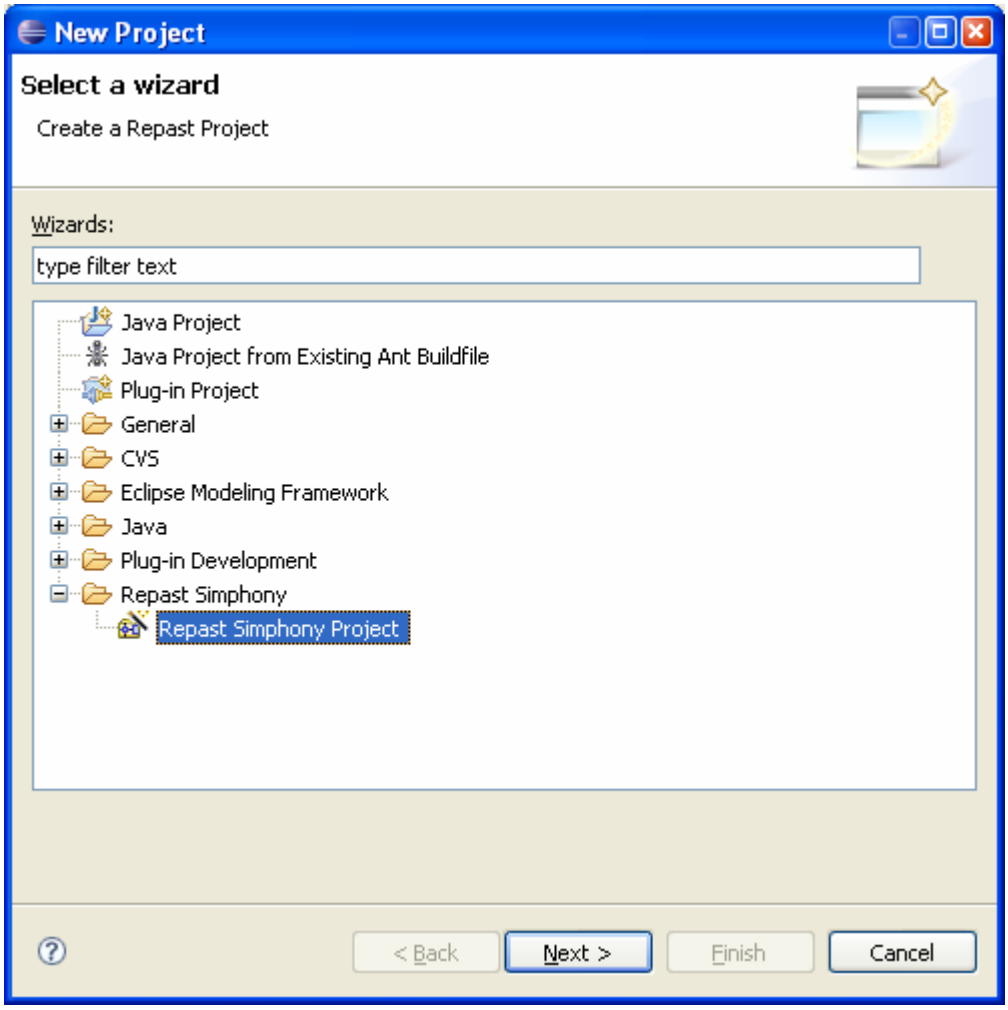

FIGURE 1 New project display wizard with Repast Simphony Project selected

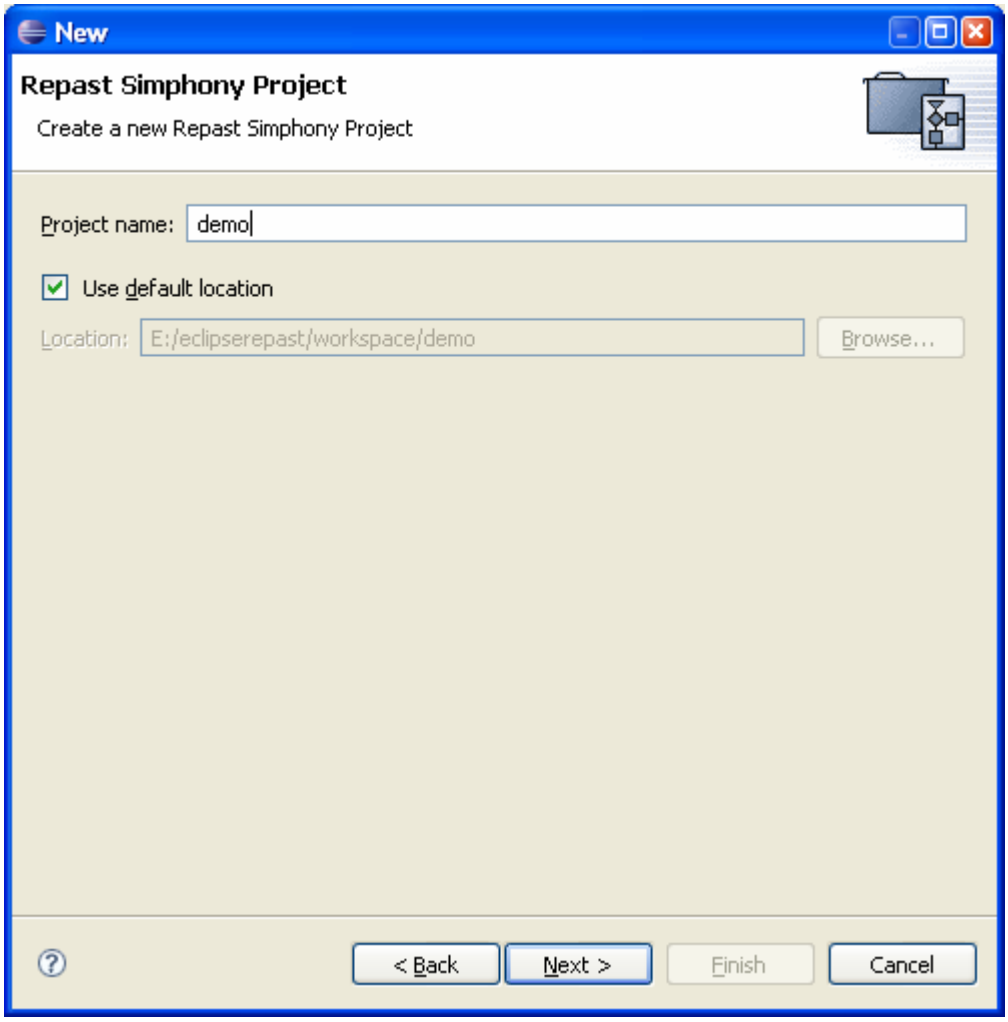

FIGURE 2 Repast Simphony Project new project basic options

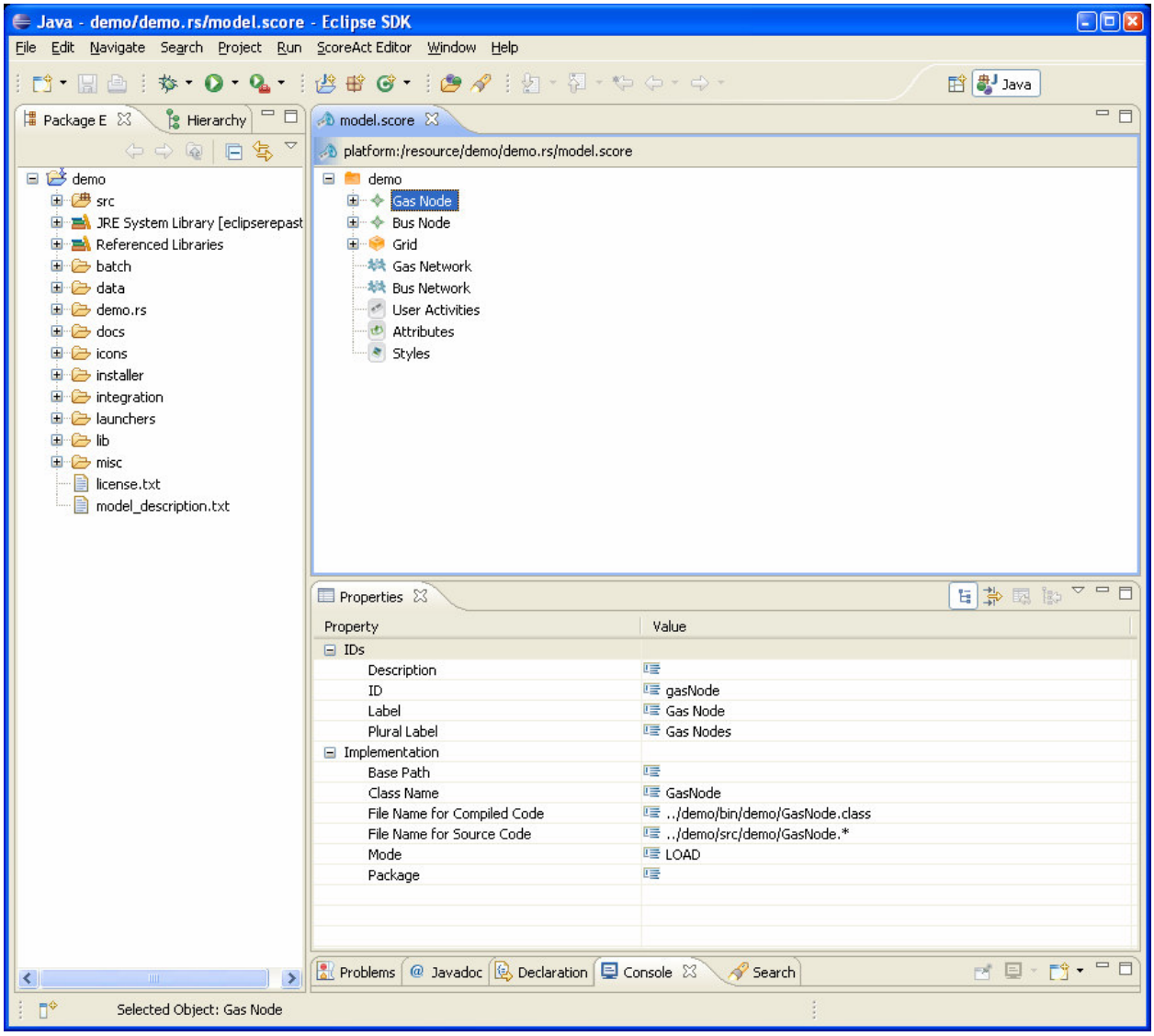

FIGURE 3 Repast Simphony project workspace showing Score editor view

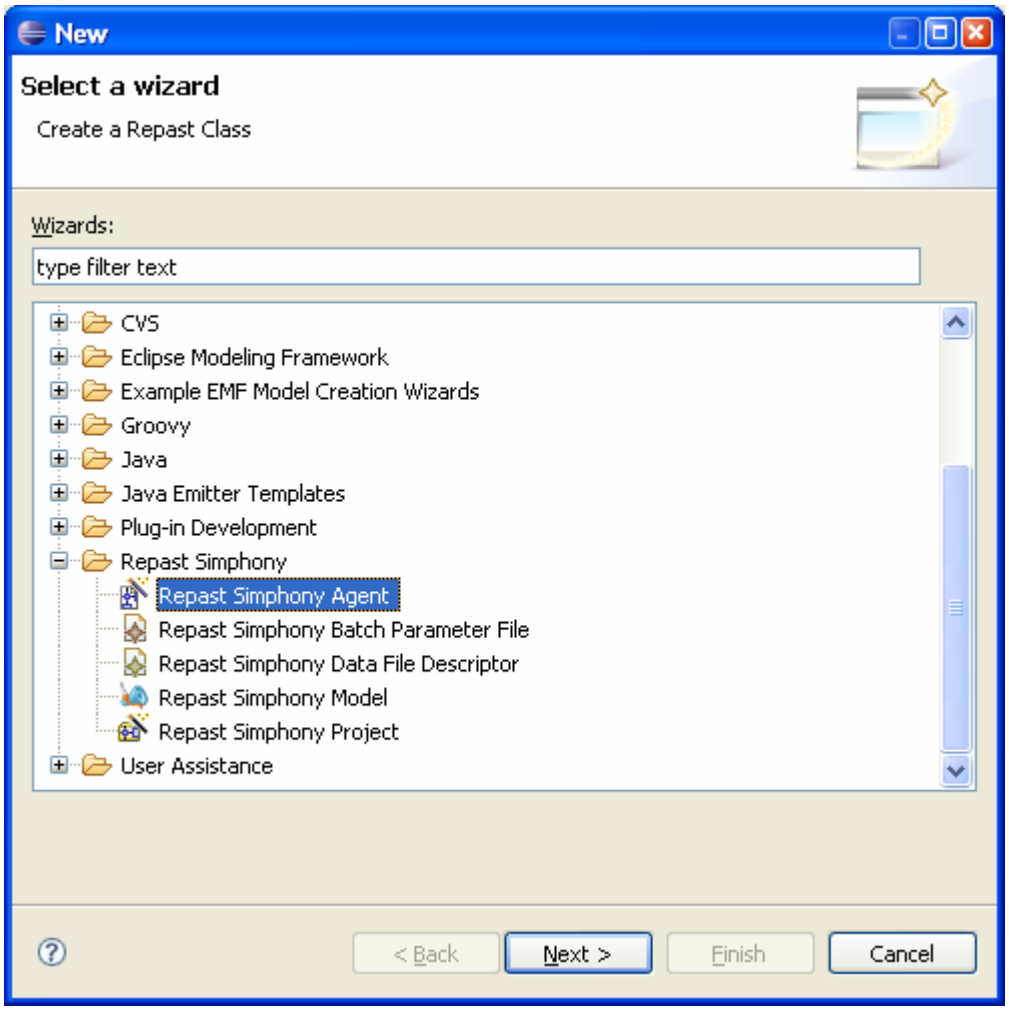

FIGURE 4 New agent creation wizard

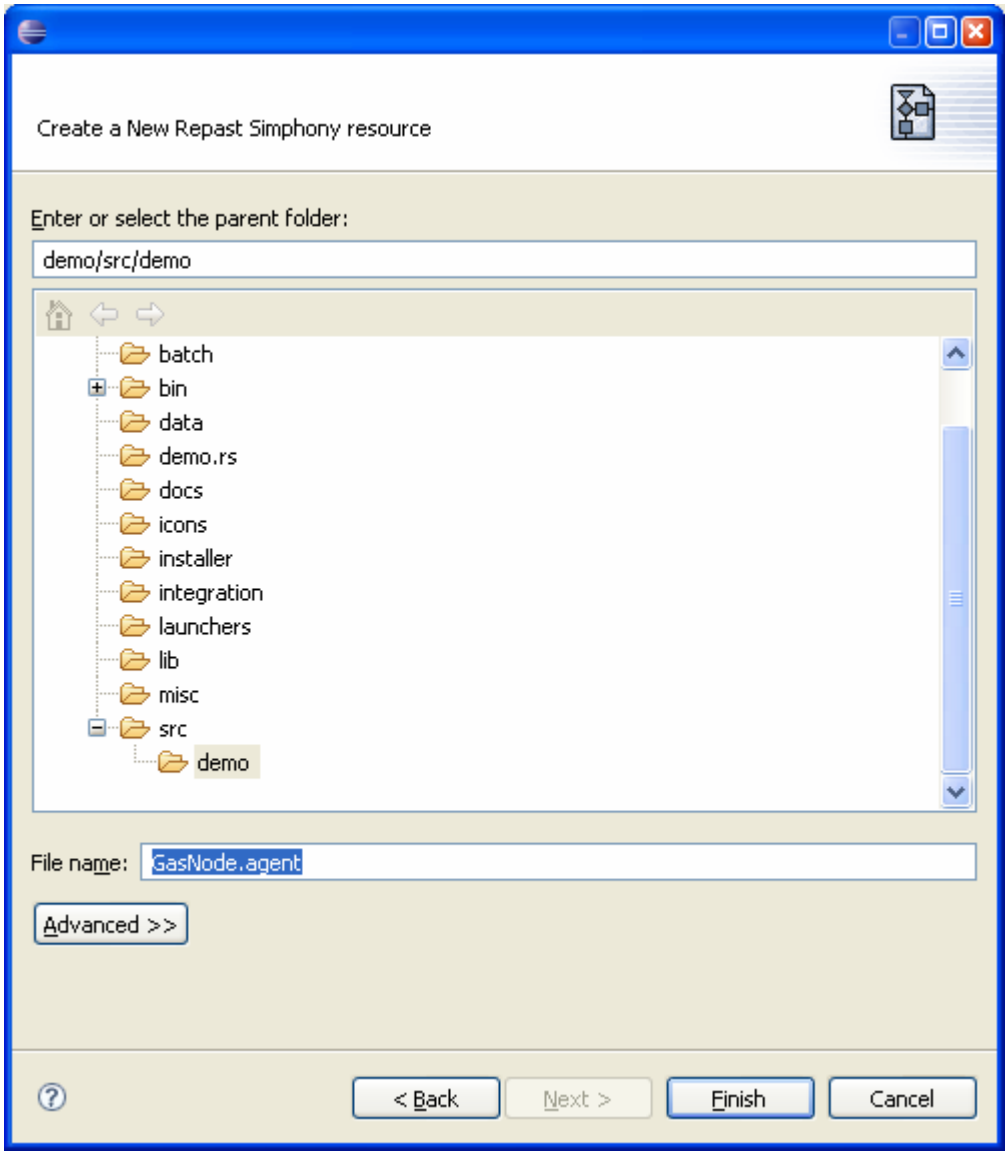

FIGURE 5 New agent creation wizard name option

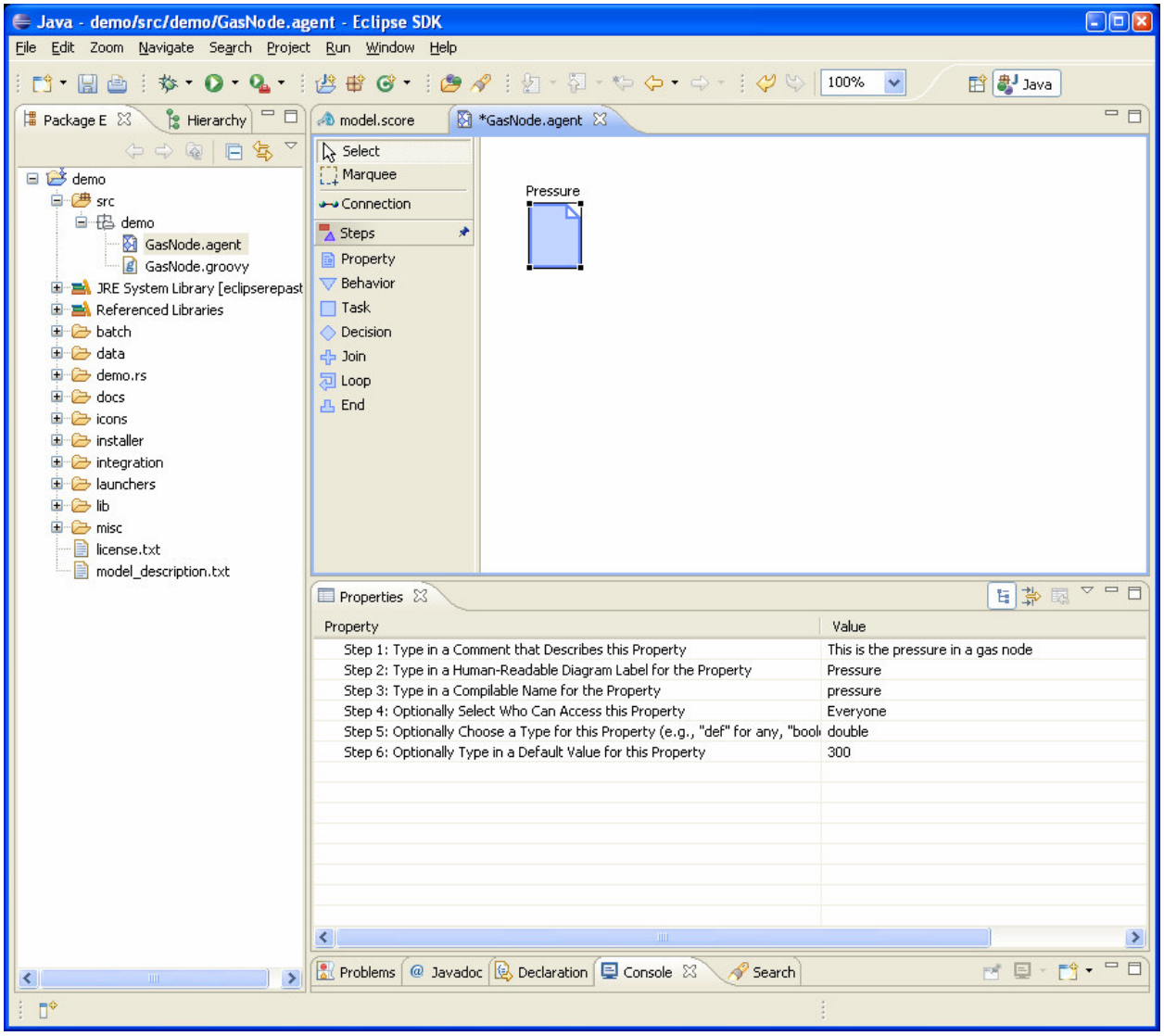

FIGURE 6 Repast agent editor showing creation of an agent property

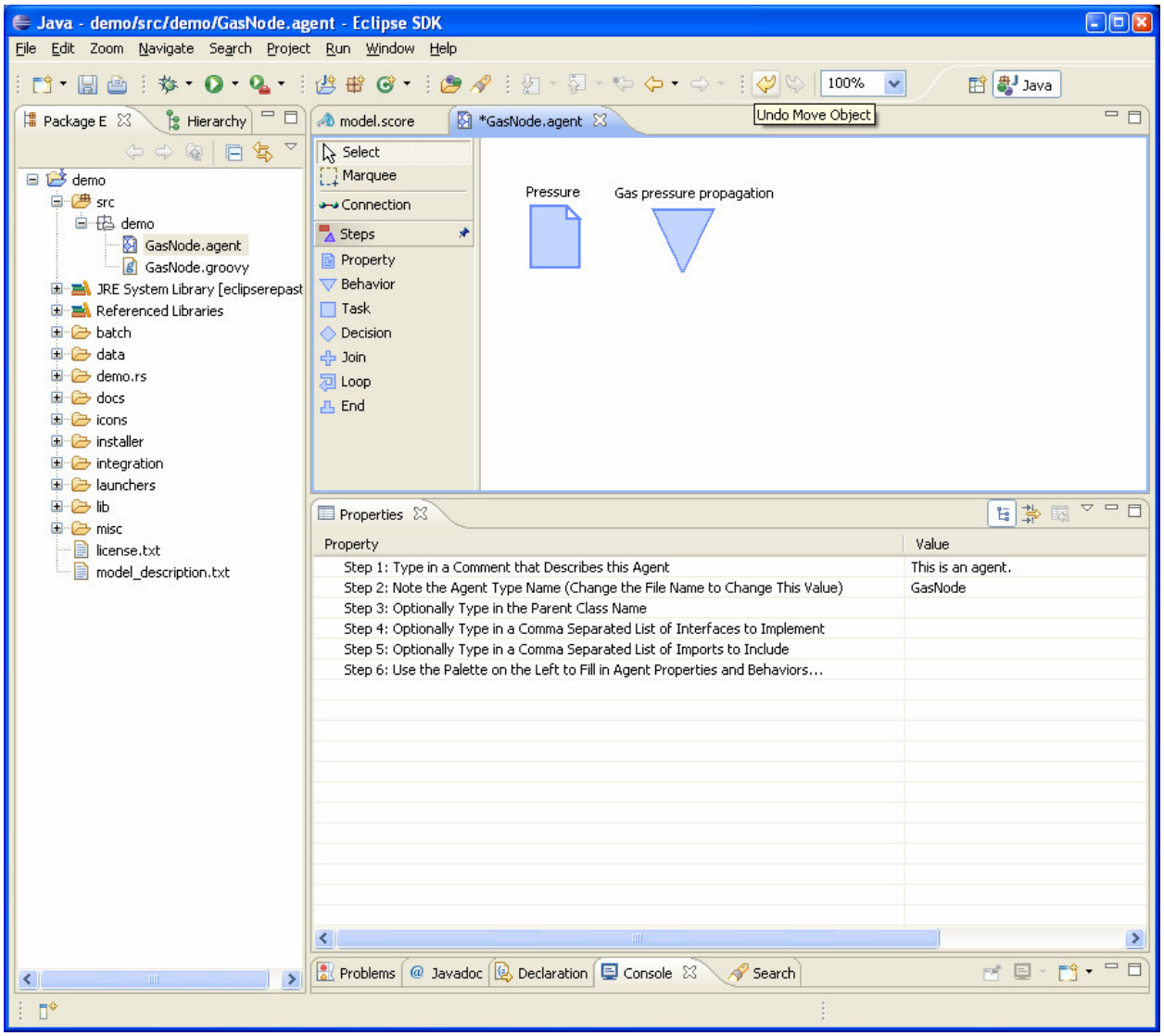

FIGURE 7 Repast agent editor showing creation of an agent behavior

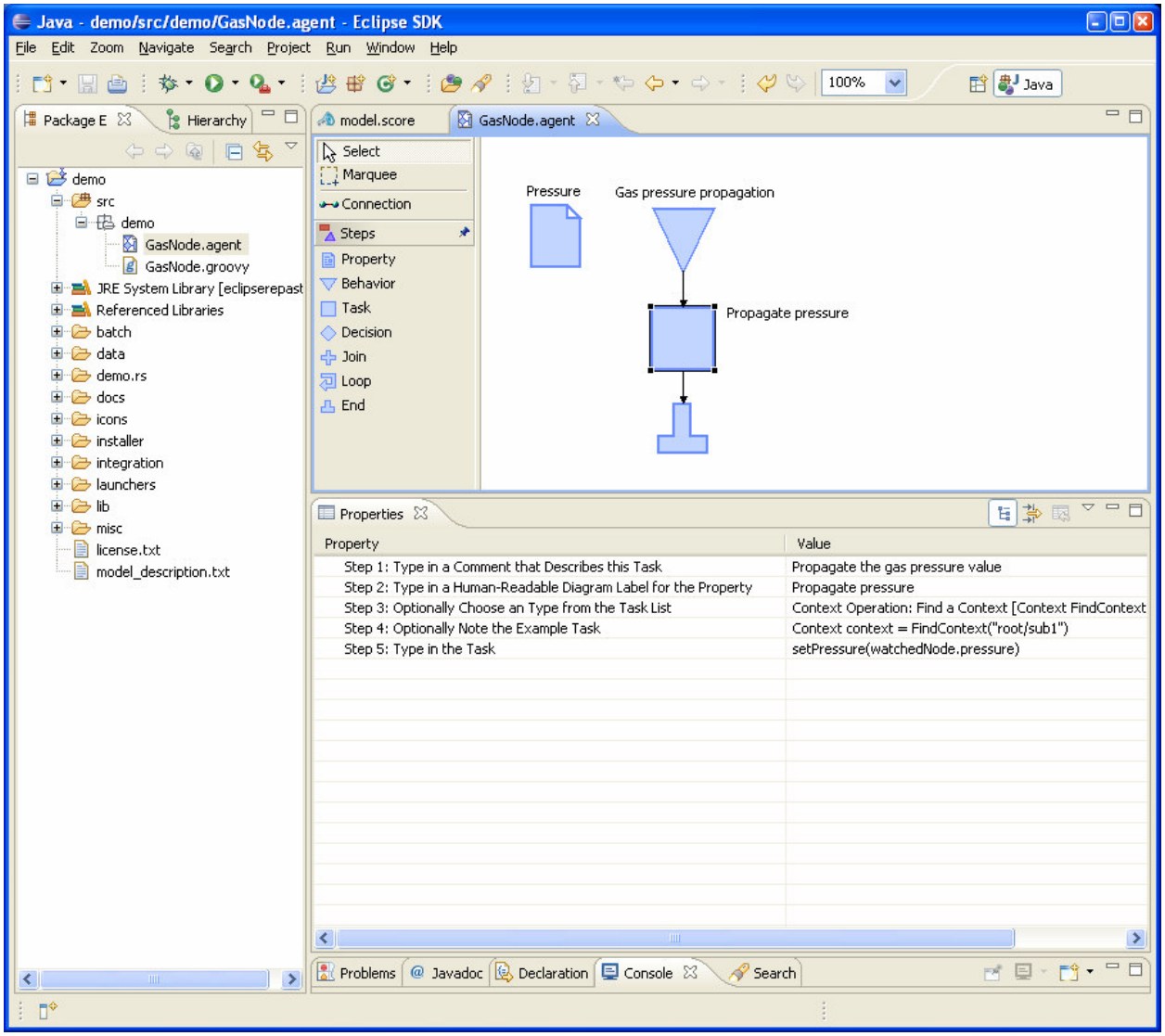

FIGURE 8 Repast agent editor showing creation of an agent task and its linkage with behavior

RUNTIME AGENT CREATION AND DISPLAY DESIGN

After the agents have been created in the Repast agent editor, the runtime may be started from the Repast IDE. Creation of user-specified data collection, output, and display may be performed through wizards in the Repast S runtime. The runtime window contains a scenario tree with contexts that branch from the main model context. The user may access each of the underlying wizards by selecting any component in the tree. Generally, one would create the components in the order of dependency: displays first, followed by data collection and data output components. The runtime graphical user interface (GUI) elements have been discussed in detail by Tatara et al. (2007a), and only the visualization elements will be discussed here.

Displays for two and three-dimensional spatial projections are created by selecting the Displays branch in the scenario tree. The user must specify at least one spatial projection and optionally one or more network projections or value layers. This demo uses a single two-dimensional (2D) grid projection on which the physical network elements are placed. The appearance of the agents is fully customizable and may be specified by the Agent Style wizard in a display item as shown in Figure 9.

The network style editor (Figure 10) allows the user to specify how the network links are visualized, including the line style, color, and width. Additionally, the line width and color may be optionally specified by the agent properties. For example, a high-pressure node may be dark blue in color and a low-pressure node may be light blue.

Additionally, the agent style may be defined by the agent style editor shown in Figure 11. The agent style editor provides options in addition to color and size, such as 2D shape, label data, and label font properties. The agent icon size may also be scaled according to a user-selected agent property. Options for 3D styles include the ability to set the 3D shape, wrap texture maps around the 3D shape, and load third-party 3D model files.

At this point, although the agent display styles have been created, no agent instances exist yet in the runtime, and thus there is nothing to visualize. The user has several options in creating agent instances, including loading from delimited or database files or by using the runtime Repast agent editor shown in Figure 12. The Repast agent editor provides the flexibility and power to create agents and arrange agents in the projections defined by the model. Agents may be created, cloned, and deleted, and agents in the projection may be freely arranged in space. The agent editor uses the styles defined for the display so that each agent type may be easily distinguished from one another.

If the model contains network projection, as with this demo, the agent editor may be used to connect the agents in the network by dragging connections between agents (Figure 13). Multiple network types are supported, and the network being edited is selected from a drop-down box. The network connections are styled on the basis of the specified styles for the display. In the case where a 3D display is to be edited, the agent editor tool will re-project the 3D display onto multiple 2D displays for editing. Figure 14 shows the connected infrastructure networks in 2D and 3D projections in the Repast runtime display.

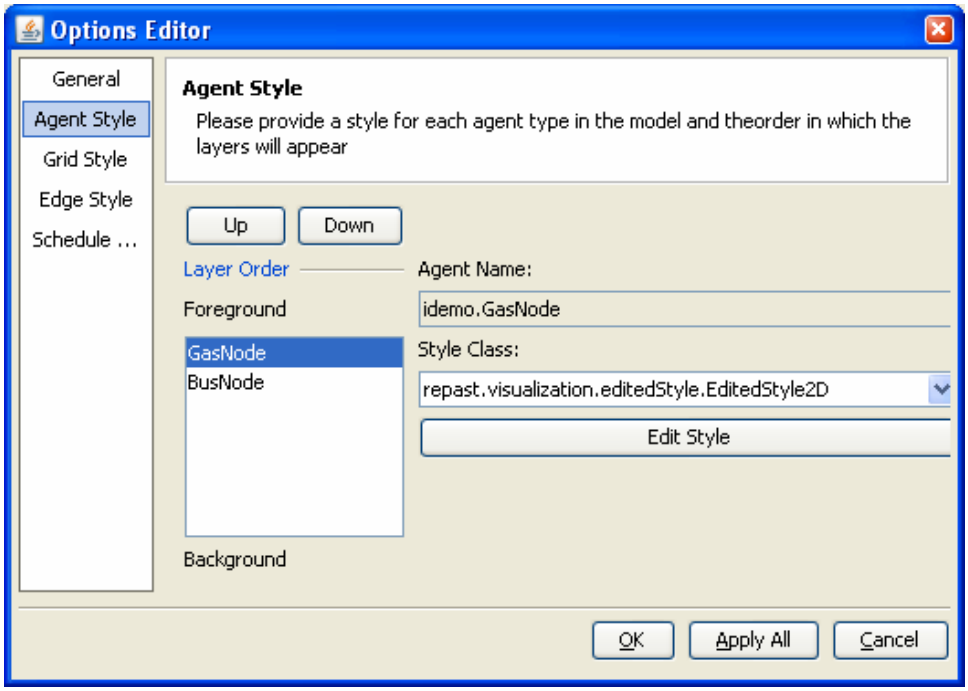

FIGURE 9 Runtime display creation wizard

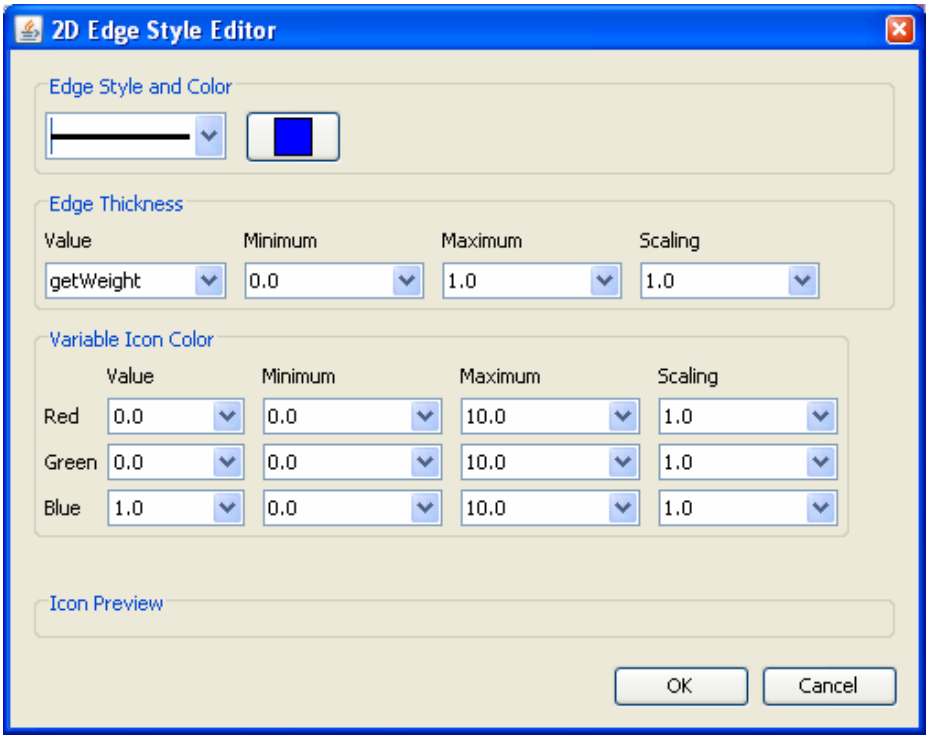

FIGURE 10 Network display style editor tool

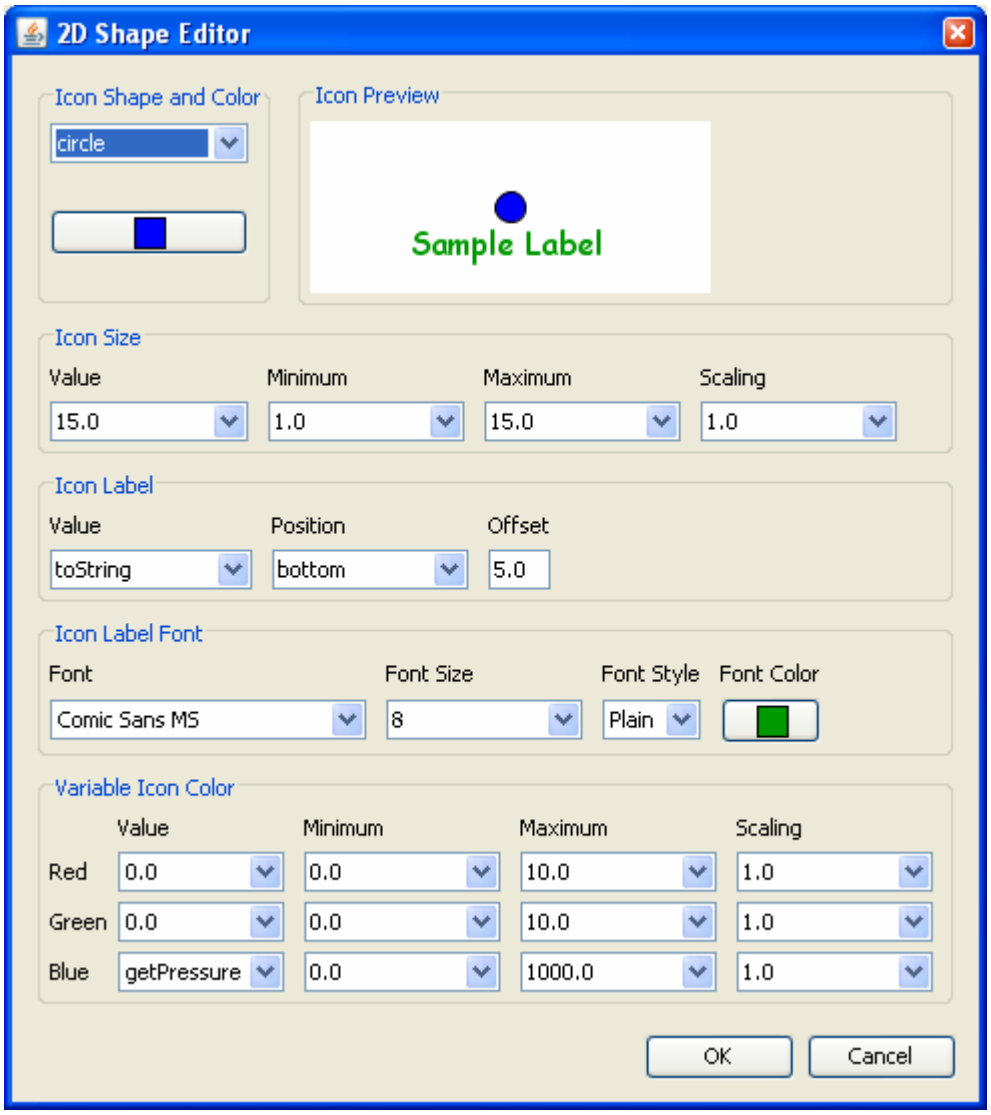

FIGURE 11 Agent display style editor tool

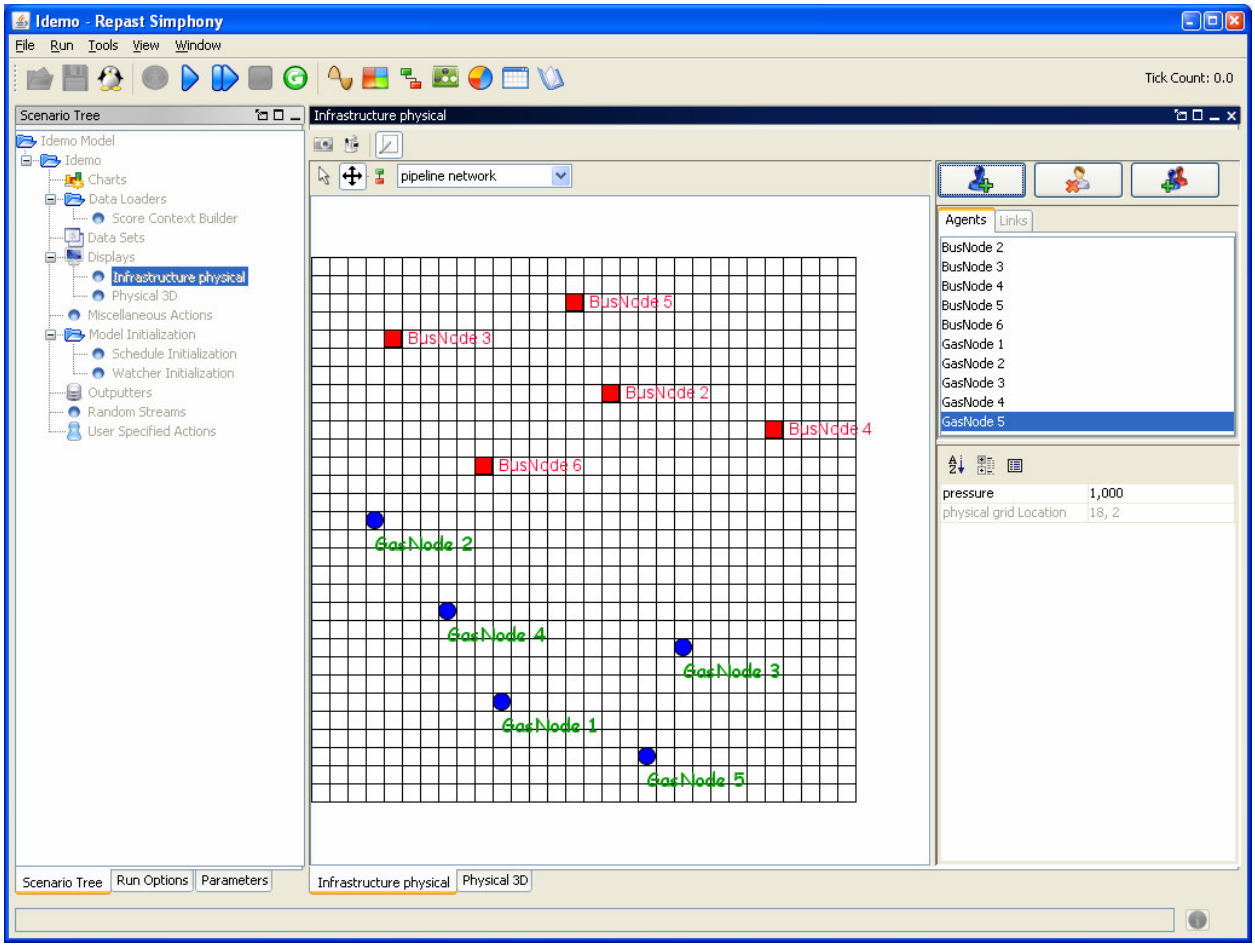

FIGURE 12 Runtime agent instance editor

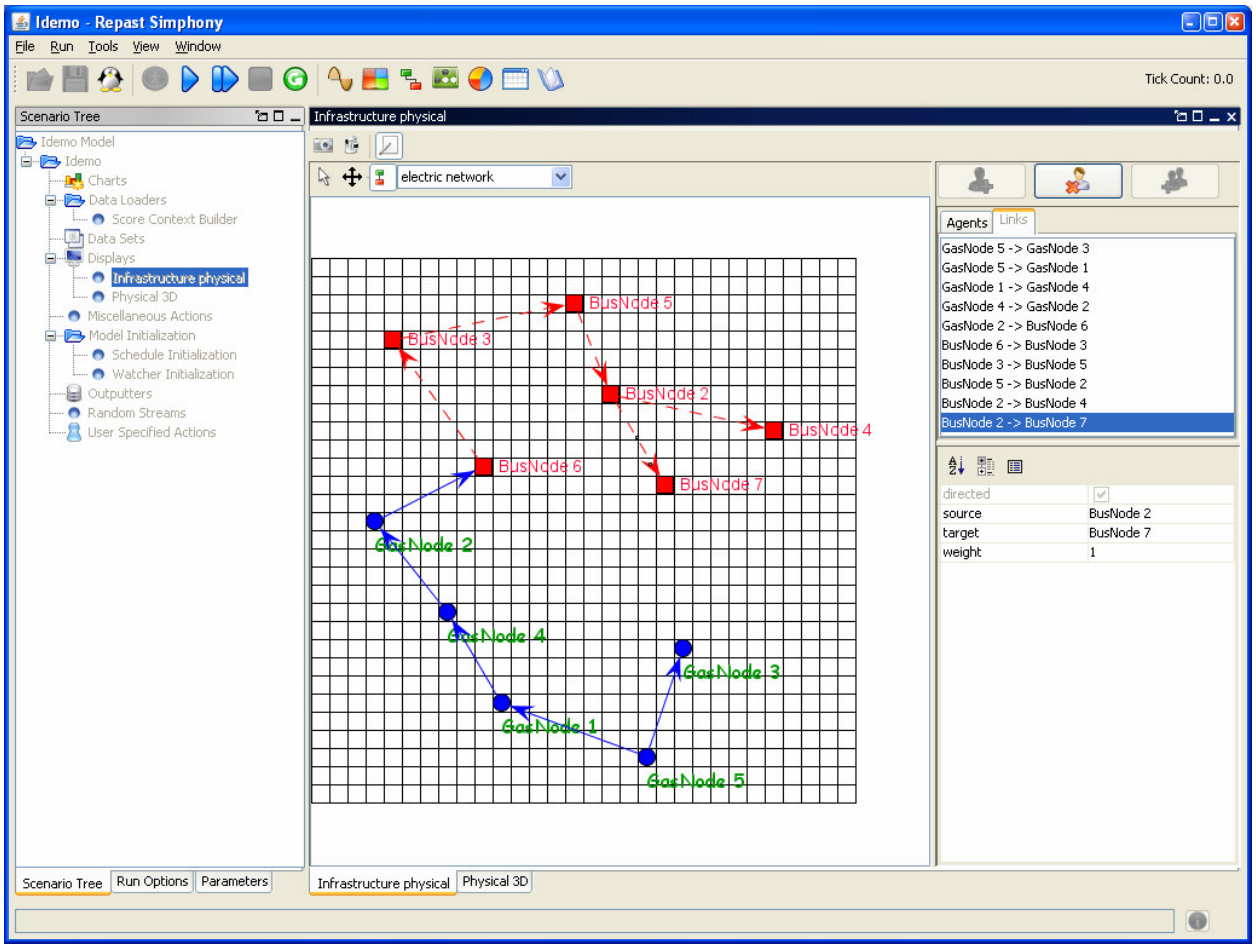

FIGURE 13 Runtime agent instance editor with network links

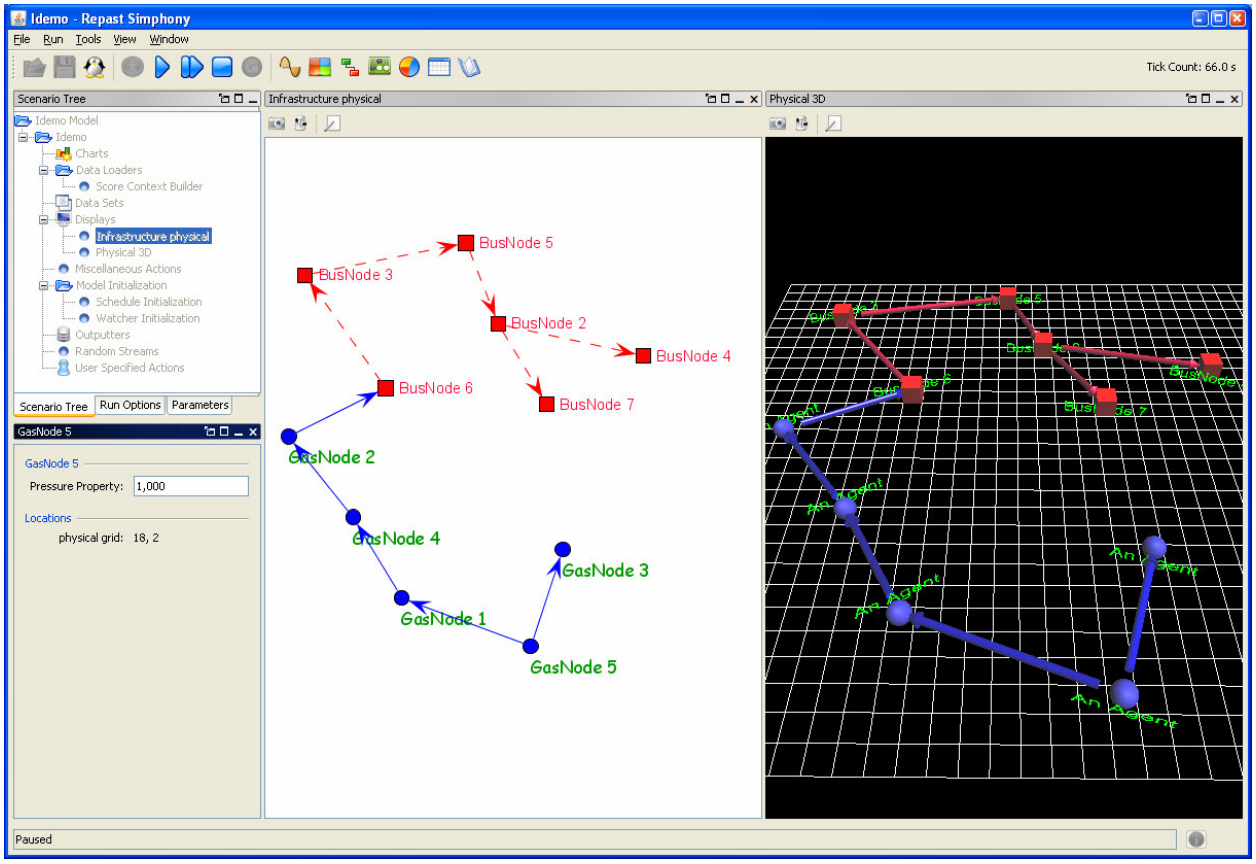

FIGURE 14 Runtime displays showing 2D and 3D visualization of networks

CONCLUSIONS

The Repast S runtime is a pure Java extension of the existing Repast portfolio. Repast S extends the Repast portfolio by offering a new approach to simulation development and execution. The Repast S development environment is expected to include advanced features for agent behavioral specification and dynamic model self-assembly. Any plain old Java object can be a Repast S model component. This paper presents an introductory tutorial and illustration of the visual modeling capabilities of Repast S by using a simple model of interconnected physical infrastructure networks.

ACKNOWLEDGMENT

The authors wish to thank David L. Sallach for his visionary leadership in founding the Repast project and Charles M. Macal for his sustaining involvement in the project. Also, the authors wish to thank Alexander Greif for contributing Flow4J-Eclipse to the software development community. This work is supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

REFERENCES

Greif, A., 2006, *Flow4J-Eclipse Home Page*; available at http://flow4jeclipse.sourceforge.net/.

Howe, T.R., N.T. Collier, M.J. North, M.T. Parker, and J.R. Vos, 2006, "Containing Agents: Contexts, Projections, and Agents" in D. Sallach, C.M. Macal, and M.J. North (eds.), *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*, ANL/DIS-06-7, co-sponsored by Argonne National Laboratory and The University of Chicago, September 21–23.

North, M.J., T.R. Howe, N.T. Collier, and J.R. Vos, 2005a, "The Repast Simphony Development Environment," in C.M. Macal, M.J. North, and D. Sallach (eds.), *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago, Oct. 13–15.

North, M.J., T.R. Howe, N.T. Collier, and J.R. Vos, 2005b, "Repast Simphony Runtime System," in C.M. Macal, M.J. North, and D. Sallach (eds.), *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago, Oct. 13–15.

North, M.J., N.T. Collier, and J.R. Vos, 2006, "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit," *ACM Transactions on Modeling and Computer Simulation* 16(1):125, ACM (January): New York, NY.

Ozik, J., M.J. North, D.L. Sallach, and J.W. Panici, 2007, "ROAD Map: Transforming and Extending Repast with Groovy," *Proceedings of the Agent 2007 Conference on Complex*

Interaction and Social Emergence, co-sponsored by Argonne National Laboratory and Northwestern University, Nov. 15–17.

Parker, M.T., T.R. Howe, M.J. North, N.T. Collier, and J.R. Vos, 2006, "Agent-Based Meta-Models," in D. Sallach, C.M. Macal, and M.J. North (eds.), *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*, ANL/DIS-06-7, co-sponsored by Argonne National Laboratory and The University of Chicago, September 21–23.

ROAD (Repast Organization for Architecture and Design), 2005, *Repast Home Page*, Chicago, IL; available at http://repast.sourceforge.net.

Tatara, E., M.J. North, T.R. Howe, N.T. Collier, and J.R. Vos, 2006, "An Introduction to Repast Simphony Modeling Using A Simple Predator-Prey Example," in D. Sallach, C.M. Macal, and M.J. North (eds.), *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*, ANL/DIS-06-7, co-sponsored by Argonne National Laboratory and The University of Chicago, September 21–23.

Tatara, E., M.J. North, T.R. Howe, N.T. Collier, and M.T. Parker, 2007a, "Building Models in Repast Simphony: A Predator-Prey Example," *Proceedings of the North American Association for Computational Social and Organizational Sciences 2007 Conference*, June 7–9, 2007, Emory University, Atlanta, Georgia.

Tatara, E., M.J. North, J. Dolph, J. Kavicky, and E. Portante, 2007b, "The Simphony Integrated Simulation Framework for Infrastructure Interdependency Modeling," *AIChE Annual Meeting*, Salt Lake City, UT, November 4–9, 2007.