Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

FINDING A SUBSTRATE FOR FEDERATED COMPONENTS ON THE WEB

John A. Miller
Andrew F. Seila
Junxiu Tao

Computer Science and MIS Departments
University of Georgia
Athens, GA 30602, U.S.A.

ABSTRACT

Recent developments in software component technology have renewed the promise of reusable software. Combining this with the possibilities of sharing simulation results and models using the Internet makes these new developments all the more important, particularly for Web-Based Simulation. Interoperability standards and data interchanges standards (e.g., XML) help facilitate having simulation models interact with other simulation models as well as other information technology components. This paper examines newer component technologies such as Enterprise Java Beans (EJB) and Jini in a search for an ideal substrate for Web-Based Simulation. Components will need distributed capabilities as well as the ability to flexibly and dynamically join an existing group of interacting components (referred to as a federation).

1 INTRODUCTION

This paper has a simple goal: to find a suitable substrate for Web-Based Simulation. Early Web-Based Simulation systems used Common Gateway Interface (CGI) Scripts and then Java Applets. Both provided remote access via the Web to simulation models. Research over the last few years is turning Web-Based Simulation into much more than just remote access: *Component technology* allows simulation elements and support modules to be rapidly combined together (e.g., Java Beans provides a base component capability). *Distribution* allows components to run on different machines on the Web (e.g., Remote Method Invocation (RMI) provides a basic mechanism for components to make remote method calls). *Federation* allows independently developed components to dynamically join an existing federation of components. Technologies that are useful for this include agent-based technologies as well as some distributed component technologies such as Enterprise Java Beans and Jini. These technologies have been tested using the JSIM Web-Based Simulation Environment (papers: Nair et al. (1996), Miller et al. (1997), Miller et al. (1998), Seila et al. (1999), Miller et al. (2000)), (theses: Nair (1997), Zhang (1997), Zhao (1997), Ge (1998), Xiang (1999), Tao (2000), Huang (2000)).

2 COMPONENTS ON THE WEB

Today, software that is large in size and provides substantial functionality is produced every day. On the other hand, users needing only a couple of advanced features are forced to purchase the more expensive product with superfluous features. To solve this crisis, component technology is a good solution. Software components are reusable building blocks for constructing software systems, and have the capability to function both independently and interactively by working with other components, Cassady-Dorion et al. (1997). By following some agreement, developers can reuse the components that were produced by other developers at different times and places.

The title of this section indicates what is needed. First a useful component technology is needed. This technology should allow software to be developed as separate modules that work together as an application or system.

This concept can be examined from different viewpoints.

- From a binding perspective, **when** are the components bound together: at compile-time, load-time or run-time? Compile-time binding is clearly too inflexible for today's Web applications. Ideally, a running application or system should be able to interact with components as needed, so that run-time binding should be supported. Use of introspection/reflection allows an application to interact with components that possibly only came into existence after the application had already started.
- From a development perspective, **how** easy is it to bring components together to accomplish something useful? Is it as simple as "plug-and-play"? What ensures that when played the components

play music and not dissonance? What orchestrates their play (builder tools, monitors, agent brokers, etc.)? How should the components interact, by method calls, synchronous remote method calls, asynchronous remote method calls, http request/reply messages, events, distributed events, etc? Which approach is the most flexible and makes it easy for developers to assemble components?

- From a distribution perspective, **where** can the components reside when they work together and can they be easily moved? Must the components be in the same process, the same machine (or virtual machine), the same local area network (or domain), or anywhere on the Web? How are the components located, by IP address and port number, by Uniform Resource Locator (URL), by name or by services offered? What type of naming or brokering service is provided? How difficult is it to move a component from one location to another?

In the rest of paper, we will address these questions by illustrating the capabilities of components in Java. Specifically, we will examine three types of components and consider their suitability as a substrate (mallable infrastructure) for Web-Based Simulation.

Table 1: Java Component Technologies

Java Technology	Environment	Distribute
Java Beans	Local Machine	None
Enterprise Java Beans	Client-Server	Basic
Jini	Distributed	Full

3 JAVA BEANS

Imagine having two Java applications (programs): one a simulation model of a FoodCourt and a second a simulation model of Emergency Room (see Figures 1 and 2).

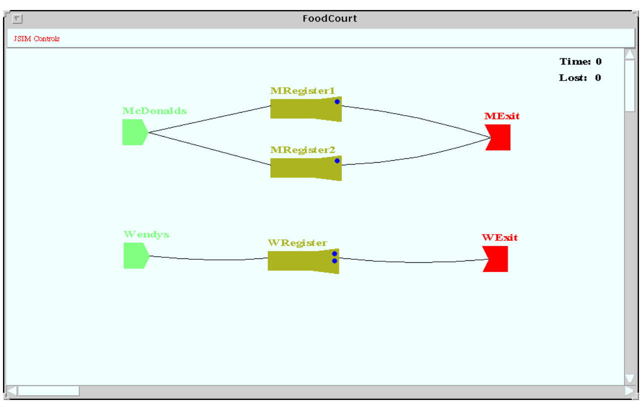

Figure 1: FoodCourt Simulation Model

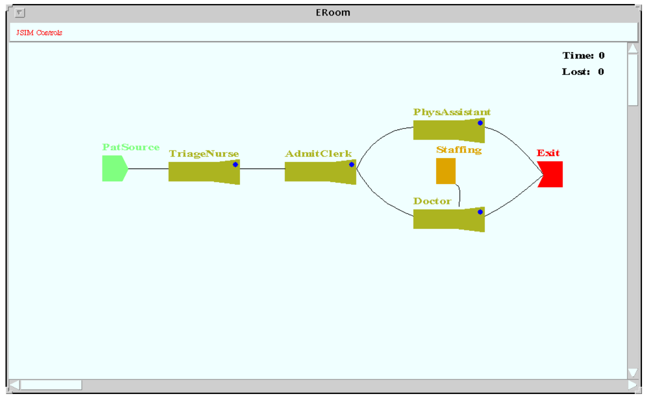

Figure 2: Emergency Room Simulation Model

Each program has a main method, so that it can be executed. What needs to be done to "upgrade" these programs to components? First, the execution of the bean needs to be initiated/controlled by a container (e.g., the Beans Development Kit (BDK) Bean Box) or some other component (e.g., a Model Agent). Next, the two models should be able to interact. This requires some form of communication. Direct communication is usually in the form of method calls, while indirect communication is usually in the form of *events*. Because of the greater flexibility offered by events, Java Beans utilizes them as the primary form of communication for beans (components). Two independently developed components can be hooked up using an event adaptor. The source component fires an event which the adaptor listens for. Once the adaptor receives the event, it calls a method in the target component. Adaptors are small pieces of code that are easy to automatically generate. Most visual development tools (e.g., BDK) allow dynamic component assembly using adaptors to hookup components without requiring any hand coding. Other important capabilities include introspection and persistence. *Introspection* allows development tools or other components to discover what properties a component has as well as get and set the values of these properties. *Persistence* allows the state of beans/objects to be saved and restored. Java serialization provides this in an internal format, while externalization may provide this in an open format (e.g., XML, Huang (2000)).

Components are very useful in simulation modeling and analysis. A simulation model may be built from components, Healy and Kilgore (1998). Simulation models themselves may be treated as components that may interact in certain ways, Miller et al. (1998), Ge (1998). In addition, components may act as service providers (e.g., information management, Miller et al. (2000), Huang (2000)) or agents (e.g., model agents, Seila et al. (1999), Xiang (1999)) for simulation models. This work allows a group of models and agents to be assembled in a bean box and executed together to simulate a more complex simulation problem, as illustrated in Figure 3 in which upon leaving the food

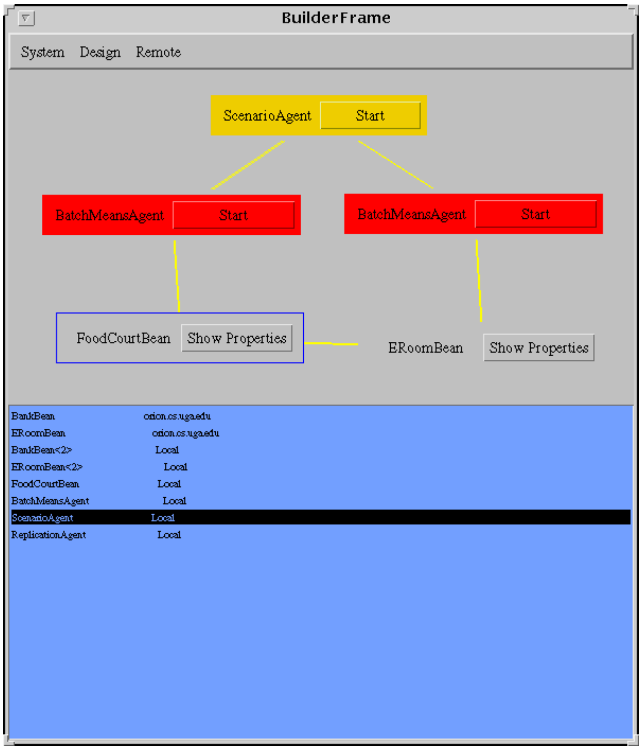

Figure 3: Interacting Simulation Components

court simulation, a fraction of the customers enter a hospital emergency room simulation.

4 ENTERPRISE JAVA BEANS

Java Beans are very useful, but they need to be taken out of the box (i.e., they simply run in a bean box (or equivalent) which executes in a single Java Virtual Machine (JVM)). The first comprehensive effort to take beans out of box is Enterprise Java Beans (EJB). The intent of EJB is to provide server-side components or beans. This complements Java Beans which provides client-side components. Although, both are components that share things in common, they have some fundamental differences and are certainly not interchangeable. One major difference is that events provide the primary means of communication for Java Beans, while remote method calls are the primary means to communicate with Enterprise Beans. Another major difference is that Java Beans are usually visible (i.e., have a GUI interface), while Enterprise Beans are not.

The design goals for the EJB component architecture are to enable enterprises to build scalable, secure, multi-platform, business-critical applications as reusable, server-side components, Roth (2000). An enterprise application model, or in other words, an Enterprise Bean is a body of code with fields and methods to implement modules of business logic, Pawlan (2000). It is a building block that can be used alone or with other Enterprise Beans to build a complete and robust thin-client multi-tiered application. An Enterprise Bean can be implemented to interact with other Enterprise Beans.

There are two types of Enterprise Beans: Session Beans and Entity Beans. By using a Session Bean, the client begins a session with an object that acts like an application, executing a unit of work on behalf of the client, possibly including multiple database transactions. By using an Entity Bean, the client accesses an object that represents a persistent entity (e.g., one stored in a database). Generally, a Session Bean represents a communication session with a client, and an Entity Bean represents data in a database, Roth (2000), Pawlan (2000).

- A Session Bean represents a transient conversation with a client, and might execute database reads and writes. A Session Bean might invoke the JDBC calls itself or it might use an Entity Bean to make the call, in which case the Session Bean is a client to the Entity Bean. A Session Bean's fields contain the state of the conversation and are transient. That means if the server or the client crashes, the Session Bean is gone.
- An Entity Bean may represent data in a database and the methods to act on that data. In an object/relational database context, there would be one bean for each object/row in a class-extent/table. The Entity Beans are transactional and long-lived. As long as the data remains in the database, the Entity Bean exists.

To simplify the programming of enterprise applications, EJB technology wraps a collection of services for supporting an EJB installation in the EJB server. These services include management of distributed transactions, management of distributed objects and distributed invocations on these objects, and some low-level system services. An EJB server provider can implement a container which provides a home for EJB components. When Enterprise Beans are installed into the container, the container provides a scalable, secure, transactional environment in which Beans can operate. The container handles the object life cycle, including creating and destroying an object. It also handles the state management of Beans.

When an Enterprise Bean is installed in a container, the container provides two interfaces to the bean: a universal interface for creating and destroying Enterprise Bean instances (its EJBHome remote interface) and an interface to the specific methods provided by the Enterprise Bean (its EJBObject remote interface). The container is also responsible for making the Bean's EJBHome interface available in the Java Naming and Directory Interface (JNDI). To construct a Bean, you must first implement the methods that are declared in the EJBObject interface.

EJB provides many advanced programming services beyond those provided by Java and Java Beans such as transaction, security, naming and deployment services. These services are all very useful in modern information systems. These are also quite a challenge for programmers to develop on their own. Because of this, EJB simplifies the development of enterprise information systems, particularly those involving Web or Application Servers. The question for the simulation community is how useful is this complex software technology for simulation modeling and analysis. In the sense that modern Web-Based Simulation systems include information technology for storage, retrieval and visualization of simulation models and results, EJB technology is likely to play a role in future Web-Based Simulation systems.

A distributed JSIM simulation system built using EJB technology has been developed supporting a small subset of the DoD's High Level Architecture (HLA), Kuhl et al. (1999), functionality, Tao (2000). Figure 4 illustrates the EJB Architecture for Distributed JSIM.

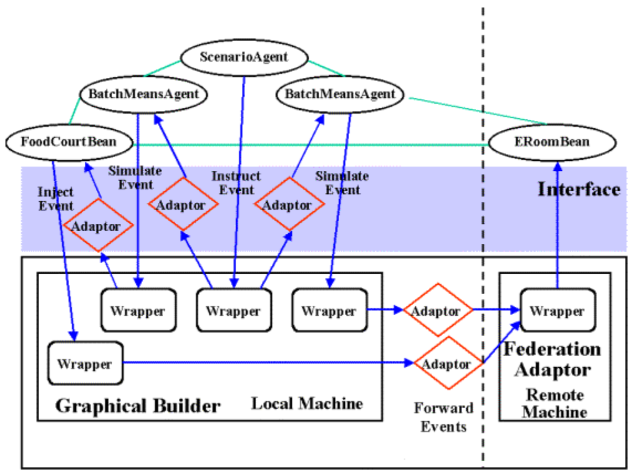

Figure 4: EJB Architecture for Distributed JSIM

At run-time, the simulation models and model agents work as federates in the the Distributed JSIM system. By functionality, Distributed JSIM consists of three parts: the Graphical Builder, the Federate Deployer and the Federation Adaptor. Among these three parts, the Graphical Builder is the most important. It acts as a client of the Federate Deployer and the Federation Adaptor, and controls their behavior. Each part of Distributed JSIM provides services for build-time (federation building), for run-time (federation execution) or for both.

Build-time services are provided by the combined efforts of the Graphical Builder and Federate Deployer. The Graphical Builder provides a graphical environment in which users can build complex federations. It can manage and display the available federates that are both from the local machine and from the remote server. The Federate Deployer is on the server side. It provides methods for managing remote federates and is implemented using EJB.

Run-time services are provided by the combined efforts of the Graphical Builder and Federation Adaptor. The Graphical Builder provides services that support federation execution, event management, object and method invocation, etc. The Federation Adaptor can store remote simulation models and control the behavior of these simulation models. It is located on the server side and communicates using RMI. Together with the Federation Adaptor, the Graphical Builder provides the communication service for linked federates, so other federates in the same federation can interact with the remote simulation model.

A basic finding of this research is that EJB can support HLA-style federated simulations in a limited and sort of convoluted way, but it is not a perfect solution, Tao (2000). It is more suitable for providing simulation services such as information management. (An early part of this work involved building a custom Bean Box in which adaptors could make remote method calls using RMI to give Java Beans a distributed capability; this was not pursued since we wish to use/customize an infrastructure, not build our own.)

5 JINI

Both Java Beans and Enterprise Beans are useful for simulation. Unfortunately, they are not interchangeable (or transmutable). Another solution, where beans (components) can be dynamically placed on the client-side or server-side would be more useful in developing federated simulation systems. This would allow simulation models to be run either on the client-side or server-side. (The client-sever distinction would then become less important than the notion of whether the bean interacts with a human.)

The emerging Jini technology, Arnold et al. (1999), holds promise as a suitable substrate for federated components on the Web. Jini is oriented to general distributed systems, not just client-server systems; it is meant to strongly support plug-and-play, to support flexible component interaction as well as component mobility, and to provide distributed events. Jini will make it quick and easy for applications (and even devices) to join an impromptu networked community.

Using Jini, a component can join a federation, interact with other components and then leave the federation. This is precisely what is needed for an HLA-style simulation, Kuhl et al. (1999). A prototype implementation of a new federated JSIM is currently underway to test the feasibility of this approach. This project will unfold in two steps.

The first step will be to allow JSIM components to communicate using distributed events. Currently, JSIM has several classes for event communication (e.g., InformEvent, InjectEvent, ReportEvent, SimulateEvent). These classes extend the JsimEvent class which extend the EventObject class. The JsimEvent would need to be modified to extend RemoteObject. JSIM also has several event listener classes (e.g., InformListener, ReportListener, InjectListener, SimulateListener). These classes extend EventListener, and would need to be modified to extend RemoteEventListener. Each class currently has a handler method such as the one shown below.

```
public void handleSimulate
                    (SimulateEvent evt);
```

All of these methods will need to be renamed `notify`. Because Jini was built to integrate well with Java Beans components, this step should be a straightforward extension of the current JSIM. At this point, an improved (over the EJB implementation) Distributed JSIM will be available.

The second step involves adding a federated nature to JSIM which will allow components to be easily added and removed from the federation. This capability currently is weakly supported since components can be dynamically assembled by a designer using the BDK Bean Box (or equivalent visual tool). Participation in the federation must be extended to distributed components, components should be mobile and joining a federation should be possible either programmatically or through an enhanced visual development/deployment tool. This work will utilize Jini's Discovery, Join, Lookup and Leasing services.

In order to make it easier for separately developed federates to communicate, the distributed events will pass XML messages between federates. This will increase the interoperable nature of the system and minimize the need for custom communication code generation or hand coding. This flexibility is gained at the price of extra processing. Federates will need to convert internal Java objects into XML and vice versa. In order for federates to understand the XML messages sent to them, we will explore tagging conventions such as Beans Markup Language (BML) or analogs of the Knowledge Query and Manipulation Language (KQML).

Figure 5 shows the XML event messages sent from federate to federate.

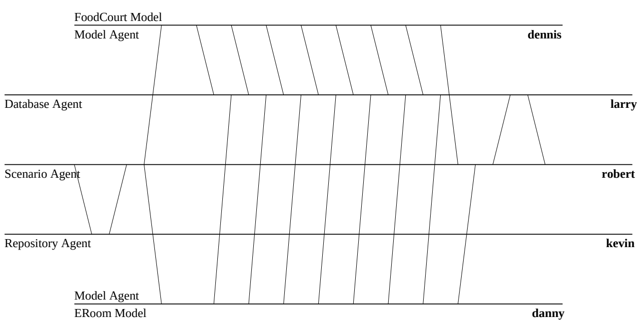

Figure 5: XML Event Messages in Federated JSIM

In this figure, a scenario agent contacts a repository agent to find simulation models and model agents suitable for carrying out the wishes of the user. Once found, they are deployed to separate machines (dennis and danny). Each model agent oversees the execution of its model. Periodically, the model agent will save information in the database. After the model/model agent complete, the scenario agent will be notified, and then the scenario agent is able to examine results stored in the database. The database agent will, as all federates do, receive XML messages. To store this information several approaches, Huang (2000), may be used: (1) use an XML database (future option), (2) convert an XML document into a Java object and directly store this in a Java-enabled object-relational database system (e.g., Cloudscape), or (3) convert the XML document into relational tables using a utility such as Oracle's XML-SQL utility, or (4) store the XML as a CLOB and extract meta-data to store in tables for searching purposes.

6 CONCLUSIONS

The search for an suitable substrate indicates that components technologies for the Web are progressing rapidly. In the JSIM project, a prototype has been built using Java Beans which works fine on a single machine. A new prototype built using Enterprise Java Beans shows enhanced capabilities. Unfortunately, the strong client-server orientation limited the flexibility of what we could do with components. It is quite helpful though in providing server-side simulation support functions (e.g., information management). The latest project which is currently underway uses the Jini distributed component technology. This technology shows great promise because of its high flexibility.

REFERENCES

Arnold, K., O'Sullivan, B., Scheifler, R., Waldo, J. and Wollrath, A. 1999. *The Jini specification*. Addison-Wesley.

Cassady-Dorion, L., Brumbaugh, M., Maheshwari, S., Wright. J., Brookshier, D., Last, B. and Mathis, J. 1997. *Industrial strength Java*. New Riders.

Ge, Y. 1998. Development of a web-based simulation environment using Java Bean. Masters Thesis, The University of Georgia.

Healy, K.J., Kilgore, R.A. 1998) Introduction to Silk and Java-based simulation. *Proceedings of the 1998 Winter Simulation Conference*, 327-334.

Huang, X. 2000. XML databases and their application to JSIM. Masters Thesis, The University of Georgia.

Kuhl, F., Weatherly, R,. Dahmann, J. 1999. *Creating computer simulation system: An introduction to the high level architecture*. Prentice Hall.

Miller, J., Seila, A., Xiang, X. 2000. The JSIM web-based simulation environment. *Future Generation Computing System Journal*: Special Issue on Web-Based Modeling and Simulation. (to appear).

Miller, J., Ge, Y. and Tao, J. 1998. Component-based simulation environments: JSIM as a case study using Java Beans. *Proceedings of the 1998 Winter Simulation Conference*, 373-381.

Miller, J., Nair, R., Zhang, Z., and Zhao, H. 1997. JSIM: A Java-based simulation andaAnimation environment. *Proceedings of the 30th Annual Simulation Symposium*, 31-42.

Nair, R. 1997. JSIM: A Java-based query driven simulation and animation environment. Masters Thesis, The University of Georgia.

Nair, R., Miller, J., and Zhang, Z. 1996. A Java-based query driven simulation environment. *Proceedings of the 1996 Winter Simulation Conference*, 786-793

Pawlan, M. 2000. Enterprise Java Beans working with entity and session Beans. `<http://developer.java.sun.com/developer/technicalArticles/Beans/EJBEntity/index.html>`.

Roth, B. 2000. An introduction to enterprise Java Beans technology. `<http://developer.java.sun.com/developer/technicalArticles/Beans/IntroEJB/index.html>`.

Seila, A.F. and Miller, J.A. 1999. Scenario management in web-based simulation. *Proceedings of the 1999 Winter Simulation Conference*, 1430-1437.

Tao J. 2000. HLA-compliant distributed JSIM. Masters Thesis, University of Georgia.

Xiang, X. 1999. Use of Agents to control the execution of Simulation Components. Masters Thesis, The University of Georgia.

Zhang, Z. 1997. A Java-based simulation and animation environment: JSIM's foundation library, Masters Thesis, The University of Georgia.

Zhao, H. 1997. A graphical designer For JSIM. Masters Thesis, The University of Georgia.

AUTHOR BIOGRAPHIES

JOHN A. MILLER is a Professor and the Graduate Coordinator in the Department of Computer Science at the University of Georgia. His research interests include database systems, simulation and workflow as well as parallel and distributed systems. Dr. Miller received the B.S. degree in Applied Mathematics from Northwestern University in 1980 and the M.S. and Ph.D. in Information and Computer Science from the Georgia Institute of Technology in 1982 and 1986, respectively. During his undergraduate education, he worked as a programmer at the Princeton Plasma Physics Laboratory. Dr. Miller is the author of over 60 technical papers in the areas of database, simulation and workflow. He has been active in the organizational committees of research conferences in all three areas, including IEA/AIE, ANSS, RIDE, WEBSIM and WSC. He is an Associate Editor for ACM Transactions on Modeling and Computer Simulation and IEEE Transactions on Systems, Man and Cybernetics as well as a Guest Editor for the International Journal in Computer Simulation and IEEE Potentials.

ANDREW F. SEILA is an Associate Professor in the Terry College of Business at the University of Georgia, Athens, Georgia. He received his B.S. in Physics and Ph.D. in Operations Research and Systems Analysis, both from the University of North Carolina at Chapel Hill. Prior to joining the faculty of the University of Georgia, he was a Member of Technical Staff at the Bell Laboratories in Holmdel, New Jersey. His research interests include all areas of simulation modeling and analysis, especially output analysis. He has been actively involved in the Winter Simulation Conference since 1977, and served as Program Chair for the 1994 conference.

JUNXIU TAO is currently employed by Intergraph Corporation, Huntsville, Alabama. She received her M.S. in Computer Science from the University of Georgia in 2000. Her research interests include simulation, distributed computing and enterprise information systems.