Building a Web−Based Federated Simulation System with Jini and XML

Xueqin Huang

John A. Miller

415 GSRC
Computer Science Department
University of Georgia
Athens, GA 30602−7404

Abstract

I*n a Web−Based federated simulation system, a group of simulation models residing on different machines attached to the Internet, called federates, collaborate with each other to accomplish a common task of simulating a complex real world system. To reduce the cost of developing and maintaining simulation models and facilitate the process of building complex collaborative simulation systems, reuse of existing simulation models and interoperability between disparate simulation models are of paramount importance. Moreover, to make such a system highly extensible, the individual federates, which could reside on the same host or physically distributed hosts, should be able to freely join and leave a federation without full knowledge of its peer federates. Simply put, an ideal simulation system should allow for quick and cheap assembly of a complex simulation out of independently developed simulations and at the same time allow the participating simulations to have maximum independence. Fortunately, this is made possible by some emerging technologies, notably Jini and the eXtensible Markup Language (XML). In this paper, we will introduce Jini and XML and present the design and prototype implementation of a Web−based federated simulation system using Jini and XML.*

1. Introduction

Developing programs to simulate complex real world systems is a challenging and expensive task. It is often desirable to reuse independently constructed pieces of a simulation and have them collaborate to solve a larger and more sophisticated simulation problem. A major effort to standardize the reuse and interoperability of simulation systems is the High Level Architecture (HLA), which was developed by the United States Department of Defense (DoD), actively supported by IEEE, and adopted by the NATO countries [Kuhl et al., 1999]. HLA defines a software architecture for composing simulation systems from components. The components are called federates and the simulation involving several federates is called a federation. The federates interact with each other through a runtime infrastructure (RTI), which offers services in six areas, including federation management, declaration management, object management, ownership management, time management, and data distribution management. HLA is an architecture, not an implementation. An HLA system can be built using CORBA or other distributed computing technologies.

An earlier effort to implement a subset of the HLA services for the JSIM Simulation Environment ([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]) was accomplished by using Enterprise JavaBeans (EJB) [Roth, 2000]. EJB is a server−side component model for the Java platform, which is designed to "enable enterprises to build scalable, secure, multi−platform, business−critical applications as reusable, server−side components" [Roth, 2000]. EJB takes the Java paradigm of platform independence a step further to independence from various legacy infrastructures such as messaging middleware, transaction support, naming & directory services, object protocols and relational databases. In addition, it delegates some difficult programming tasks, such as distributed transaction, distributed object invocation, security, load balancing, and connection pooling, to EJB server and container providers, greatly simplifying application development and making it an ideal model for multi−tier thin−client/thick−services business applications. It could also be very useful for managing simulation data on the database side. However, because it’s client−server orientation, EJB is not a perfect solution to an HLA−style simulation system [Miller et al., 2000].

In our search for an ideal technology to implement a fully distributed and highly extensible Web−based federated simulation system, we have found our answer in two promising Web technologies, Jini and XML. Jini is a universal network−computing model for the Java platform, and XML is a universal format for data exchange on the Web. We think Jini and XML nicely complement each other. Together, they hold great promise for future information systems, including federated simulation.

We will first introduce Jini and XML in section 2 and 3. Then, we will give an overview of JSIM and present the design and implementation of the current Java Bean component−based JSIM in section 4. In section 5, we will describe our vision of the future JSIM as a fully distributed and highly extensible Web−based federated simulation system enabled by Jini and XML. We will draw some conclusions and point to some possible future work in section 6.

2. Jini: A Universal Network Computing Model

While EJB redefines the current model under which server−side enterprise business logic is developed and deployed, Jini redefines the current model under which a client discovers, manages and communicates with the services it requires. The Jini vision is to turn the network into the client's computer by supplying the client with a federation of remote "plug and play" devices and services in a dynamic configuration (the Jini Federation) that is personalized for each client [Jini, 2000].

In architectural model, Jini supports peer−to−peer communication with variable−size client and variable−size services. On the other hand, EJB supports multi−tier thin−client and thick services. According to Jim Waldo, Sun's chief Jini architect, the Jini architecture is based on the idea of federation rather than central control [Venners, 1999]. To the client, the network is a computer made up of a federation of devices and services in the form of mobile objects or agents. In Bill Joy's words, the Java/Jini layer on which those mobile objects and agents reside can be called the BIOS (basic input/output system) of a network computer [Venners, 1999]. However, Jini is not intended to be anything like a traditional operating system, which knows about everything and controls everything. Instead, it is intended to give an object−oriented interface to the computer of the future.

A Jini system consists of the following parts [Jini Architecture Specification, 2000]:

- A set of components that provides an infrastructure for federating services in a distributed system.
- A programming model that supports and encourages the production of reliable distributed services.
- Services that can be made part of a federated Jini system and that offer functionality to any other member of the federation.

The goals of the Jini system include the following:

- Enable users to share services and resources over a network.
- Provide users easy access to resources anywhere on the network while allowing the network location of the user to change.
- Simplify the task of building, maintaining, and altering a network of devices, software, and users.

A key concept in Jini is service. A Jini service is "an entity that can be used by a person, a program, or another service" [Jini Architecture Specification, 2000]. Under the notion of a service, Jini unifies everything from the user of a system of Jini technology−enabled services/devices to the software available on the machines, and to the hardware components of the machines themselves [Jini Core, 2000]. Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Services in a Jini system communicate with each other by using a set of interfaces called a "service protocol". A lookup service maps interfaces indicating the functionality provided by a service to sets of objects, which implement the service. A lookup service may include other lookup services or contain other forms of lookup service. A pair of protocols called "discovery" and "join" are used to add a service to a lookup service.

Jini supports object/code mobility, security, lease−based service access, and transactions. Jini code mobility is accomplished by using Java Remote Method Invocation (RMI) as the underlying protocol for communication between the Jini services. RMI has extended the traditional notion of remote method call mechanism to allow both data and object to be moved around a network. Security on accessing a service is ensured through an access control list. A lease−based service requires the service user and provider to negotiate about the period of the lease and renew the lease with the lookup service if/when necessary. Because of its distributed nature, Jini supports transactions through the two−phase commit protocol as supplied in the Jini Transaction Service interface.

Another nice feature of Jini is that it provides support for distributed events, which is a natural extension to the Java Bean paradigm of event−based communication. Its purpose is to allow an object in one Java virtual machine (JVM) to register interest in the occurrence of some event occurring in an object in some other JVM to receive a notification when an event of that kind occurs [Jini Core, 2000]. Due to the fact that event delivery is inherently unreliable in a distributed system, Jini

- allows various degrees of assurance on delivery of a notification,
- supports different policies of scheduling notification, and
- explicitly allows the interposition of objects that will collect, hold, filter, and forward notifications.

The participants in a distributed event include the object that registers interest in an event, the object in which an event occurs (event generator), and the recipient of event notifications (remote event listener, which could be the object that has registered interest in the event or a third party). An object can register interest in multiple events with the same event generator or different events with different event generators. An event generator can also allow multiple objects to register interest in each of its events.

When a state change occurs in the event generator, a remote event object encapsulating the state change information is created and passed along with the notification to all registered remote event listeners. A remote event contains information about the kind of event that has occurred, a reference to the object in which the event occurred, a sequence number allowing identification of the particular instance of the event, and a handback object. The handback object is originally supplied by the event registrant to associate arbitrary information or actions with the notification. An event listener distinguishes the events through the reference to the event generator, the event identifier, and the sequence number.

3. XML: A Universal Format for Data Exchange

Described as the "Second Coming of the Web", the eXtensible Markup Language (XML) [W3C XML Site] is an emerging and rapidly involving technology that is shaping the second−generation Web, following the revolution started by the combination of hypertext and a global Internet [Bosak and Bray, 1999]. The most exciting possibility opened up by XML is the semantic Web envisioned by the creators of XML. Such a Web will be powered by systems that talk to each other in XML so that there will be no confusion about the syntax of the data being exchanged. In addition, with the help of XML−related standards, such as the XML meta−data standard, Resource Description Framework (RDF) [RDF], XML−enabled data can carry with itself not only structural information but also semantic information. This will greatly facilitate the automatic processing of information. For example, XML will allow agents roaming on the Web to pull data from the Web by semantics. The agents can then automatically extract the portion it has interest in, synthesize the information with other information it has gathered, possibly store the final results in a database, and/or forward them to other agents. In a recent talk [Berners−Lee, 2000] at the XML Open 2000, Tim Berners−Lee, Director of the World Wide Web Consortium, presented an architecture for building a trusted semantic Web on top of ontology and XML technologies, such as RDF, XML Schema [XML Schema], eXtensible Stylesheet Language (XSL) [XSL], XML Linking (XLL) [XLL], and XML Digital Signature [XML Signature].

Although XML is created as a standard format for exchange of structured data over the Web, it is likely to play an increasingly important role in other types of future information systems. Among its many merits, it has huge potential in opening up the possibility of syntactic and semantic interoperability between disparate systems. With regard to syntactic interoperability, the structure and type information about an XML document can be precisely captured in an XML schema file, which can be passed along with the XML document from one application to another application. Since XML Schema is also a standard, there will no ambiguity or misunderstanding about the syntax of the XML document. At the semantic level, the XML RDF is expected to work closely with ontology mapping to provide domain specific semantics to an XML document.

In addition, as a self−descriptive, text−based, and universal data format, XML is an excellent choice for representing data, or even objects, that need to be transported over the network between disparate systems. In fact, a preliminary specification for XML Messaging has been published as an IETF Internet Draft to allow "reliable, resilient, secure, tamper resistant, authenticated exchange of XML or other electronic documents over insecure transport mechanisms" [Cover, 2000]. An XML Protocol Working Group was created by W3C in September 2000 to develop "technologies which allow two or more peers to communicate in a distributed environment, using XML as its encapsulation language" [XML Protocol]. Currently under review are over 20 XML−based protocols [XML Protocol Comparisons].

Despite the many dreams, visions, and uses that XML seems to promise, XML in itself is just a format for encoding structured data. So how can programs access and manipulate XML−encoded data in the first place?

Currently, there are several ways to access an XML document. At the element level, two standard application programming interfaces (APIs), namely, the Simple APIs for XML (SAX) and the Document Object Model (DOM) APIs are most often used. SAX provides an event−based programming model. A SAX parser uses callbacks to report whatever it reads from an XML document back to a program. Since it takes only one scan for a SAX parser to process an XML document, SAX parsers are very efficient and good for large XML documents or stream−based XML documents. However, because of its callback mechanism, a SAX parser may not be flexible enough for those programs that need to manipulate an XML document in sophisticated ways. On the other hand, DOM provides a tree−based programming model. A DOM parser loads an entire XML document into memory and then converts it into a tree structure. The DOM APIs allow programs to traverse and modify the XML tree. DOM parsers are more flexible but less efficient than SAX parsers,.

At the object level, some efforts are being made to automatically convert DTD or XML schema into class files in an object−oriented programming language such as Java or C++. The purpose is to allow programmatic manipulation of XML documents at the object level rather than at the element level. Work in this direction includes but not limited to the Java XML Data Binding project (Sun), Dynamic XML (Object Space), Quick (JXML), and Castor (exolab.org).

For XML documents stored in databases, various tools, utilities, and APIs, either supplied by some database vendors or developed by third parties, are available today. To standardize efficient searching, extracting, and manipulation of XML documents, the W3C is actively working on the XML Query Language that will support declarative queries against a collection of XML documents. Based on the XML data model, the W3C XML Query Language is expected to incorporate XInfoset and XPath, and support XML Schema and XPointer.

The point is, XML is not just a nice data format, it is designed and developed with interoperability and ease of automatic processing of data in mind. Programs can use standard mechanisms to discover the structure, data type, or even semantics of XML−encoded data.

4. The JSIM Simulation Environment

JSIM is a simulation environment implemented in Java. The target users of JSIM are simulation analysts and simulation model developers. Our goal is to provide simulation analysts with an integrated environment in which they can visually design and create simulation models, customize simulation input at run time, execute models locally or in collaboration with remote models, and store and access simulation results. The system has been carefully designed with maximum flexibility in mind so that it can be easily extended to provide more sophisticated functions without breaking its existing functions and without modifying much code.

The current system consists of a foundation library, a visual model designer, generated models, model execution control agents, and a database agent. The foundation library contains classes supporting the execution of a simulation, including queuing, simulation event scheduling, and some statistic analysis classes. The model designer has automatic code generation capability. It allows a user to design a simulation model, enter input parameters for the model, and then generate Java code or store the model in a special format. Everything is done through pointing and clicking on a GUI canvas or entering text in dialog boxes.

The code generated for each model includes a Java bean and a Java applet, both are ready to be compiled and executed. The model applet can execute independently in an applet environment. The model bean can be loaded into a bean builder, dynamically customized, and then wired with a model agent. A model agent uses sophisticated statistical methods to control the execution of the simulation and ensure the integrity of simulation data. It also handles communication with the other agents on the behalf of the model [Seila et al., 1999] to allow further processing of the simulation data. A model bean can also be dynamically connected with other model beans to collaborate on more complex simulation tasks. In the case of a model federation, a scenario control agent is used to coordinate the execution of the overall simulation. The input and output of each model can be collected by the model agent and sent to the database agent for persistence.

We store the simulation data for each execution of a simulation as an object using Cloudscape, an embedded object−relational database management system. Our purpose is to allow for ease of use, easy maintenance, and maximum flexibility. The database supports SQL queries. The query results are returned as an XML document, which will be displayed in an XML tree viewer.

Figure 1 is a diagram of the current JSIM system architecture. The design agent is not included here because it is used separately and does not communicate with other components.

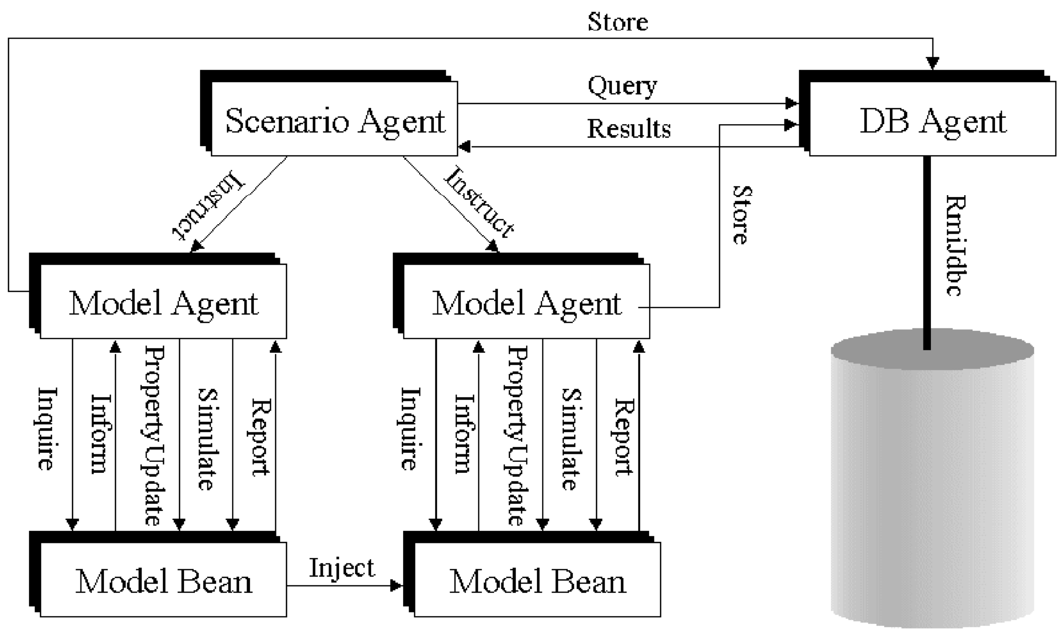

Figure-1 JSIM Architecture

The beans communicate with each other through Java Bean events. The objects passed along with the events can be normal Java objects or Java objects serialized in XML format. The choice can be made when the system is started.

Currently, the scenario agent provides an interface for a user to control both the execution of a simulation and querying of the simulation results. In the future, we plan to separate the execution and querying functions of the system. The scenario agent will still be in charge of the execution of the simulation. But its querying capability will be separated. Instead, a higher level agent, called the Query Driven Simulation Agent, will be developed. The QDS agent will provide an integrated user interface that will serve as a major entry point to the simulation system.

The system is reasonably extensible, because we have been trying very hard, throughout our design and implementation, to generalize the common functions at the top of the class hierarchies. Developers who wish to extend our system only need to add minimum customization code for their specific purposes.

5. Toward Web–Based Federated Simulation

With Java Bean events, the beans can talk only within a single JVM. As we mentioned earlier, an effort was made to extend JSIM in support of the distributed High Level Architecture using Enterprise Java Beans, but the client–server nature of EJB seems inadequate to accomplish our goal of a fully distributed simulation system. Hence, we are currently migrating toward Jini, a technology designed to support federated systems on the Internet.

Fortunately, migrating from Java Bean events to Jini distributed events is not a very difficult task. But we still have work to do to tap the full potential of Jini. For example, currently the beans have to be loaded into a bean builder to be assembled into a simulation system. This will allow a single user to have total control of the entire simulation execution. It may be desirable in some situations, and will probably continue to be supported in the future. However, this may not be possible in a distributed environment, where no one has full knowledge about available models and model agents, let alone how to communicate and collaborate with each other. There are other issues, such as unreliable event delivery, security, and distributed transaction. All these problems can be relatively easy to solve in a Jini environment. Jini provides standard interfaces for its components to discover and join a federation of services, negotiate a communication protocol and a lease with a remote object for service use, register interest in the occurrence of an event in a remote object, and receive notification that an event happened. Jini has built–in support for default semantics of security and distributed transaction. It also allows its components to extend it to redefine the meaning of security and distributed transactions.

So far, Jini seems to be a perfect architecture for building a fully distributed federated simulation system. But how much will the different simulation systems actually understand each other? How meaningful will be their communication? These questions remain largely unanswered. Our vision is to use XML to represent the data carried with the events. By doing so, we open up the possibility that JSIM could talk with other simulation systems using XML, and eventually collaborate with them on some simulation tasks in the future. By using XML, at least there will be no confusion over the structure and data type of the information being passed between the different systems.

To achieve semantic interoperability, there could be two approaches. First, the simulation community can work out a common Document Type Definition (DTD) [XML, 1998] that will define a class of XML documents for the simulation domain. This requires a DTD developer to work closely with simulation domain experts to carefully define a set of semantic tags and the structure of the simulation data. The resulting DTD should be general–purpose enough to be able to represent most simulation data. In addition, it has to be reasonably simple to handle and relatively flexible to allow future extension. It may seem unlikely to get an entire domain to agree on a single DTD. But this approach has already been adopted in some areas, such as mathematics, chemistry, and biology. Some examples of domain–specific XML–based languages are MathML, Chemical Markup Language, BioML, BSML (Bioinformatic Sequence Markup Language), the Weather Observation Markup Format, AML (Markup for Astronomy), and some graphics markup languages [Laurent and Biggar, 1999]. Another new development is, the XSL Transformation (XSLT) [XSLT] makes it possible to convert an XML document conforming to one DTD into another XML document conforming to a different DTD. This could be useful when a system has to use a DTD different from the common DTD.

A second approach toward semantic interoperability would be to combine the XML meta–data standard and ontology mapping. This approach is still under active research and development. The ontology mapping also requires involvement of domain experts to define a set of common vocabularies and their relationships in the domain. Among some examples of XML meta–data format, the XML Meta–data Interchange (XMI) is an OMG–endorsed object meta–data interchange format for exchanging objects between modeling tools. The Conceptual Knowledge Markup Language (CKML) and its subset, Ontology Markup Language (OML), provide tools for describing relationships. The Australia New Zealand Land Information Meta–data (ANZMETA) allows exchange of information describing land in Australia. The Synchronized Multimedia Integration Language (SMIL) is a W3C Recommendation describing a vocabulary for the creation of multimedia presentations [Laurent and Biggar, 1999].

Now, the big picture is clear. But what will our future Jini–and–XML simulation system look like? Figure 2 is what we have in mind.

In this figure, the scenario agent acts as a federation coordinator and a global transaction manager. Models can join the federation using the Jini discover and join protocols. Non–JSIM models can also register with the scenario agent and communicate with a peer federate to

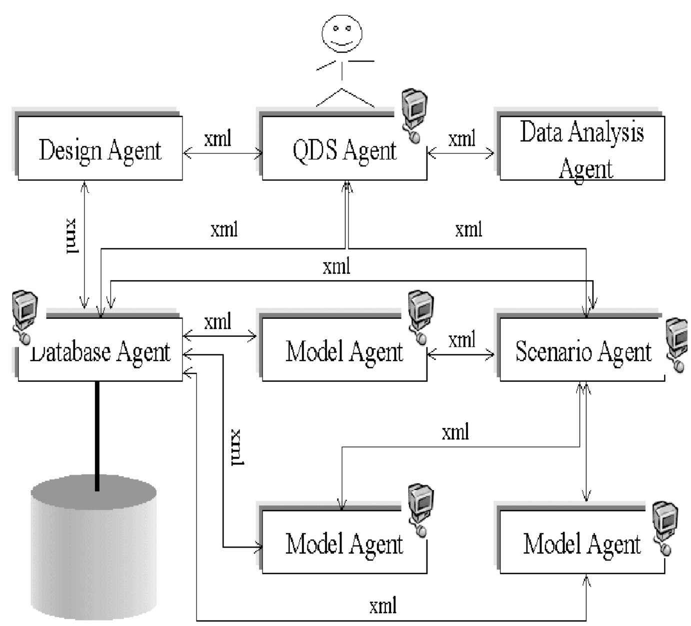

Figure-2 JSIM Federated Simulation with Jini and XML

collaborate on a common task. This is possible because the messages carried with the remote events are in XML format. As long as the participating federates share a common set of vocabulary, they should be able to understand each other reasonably well. After the simulation is done, the scenario agent can use the two–phase commit protocol to make sure that either the input and output of all federates are stored into a database or nothing is stored. A database agent can be located through the Jini look up service. There could be multiple database agents residing on different machines, or there could be one database agent that can access one or multiple databases.

Sitting at the top is the QDS Agent. We can think of it as an agent who knows how to communicate with the database agent, the model designer, and the scenario agent to assist a casual user to get things done with JSIM in an easy and orderly manner. A typical scenario of using the QDS Agent goes like this. A user browses the model repository and decides whether to query existing models or design new models. If a query on some combination of input and output of a model execution is submitted and at least one match is found, the QDS agent returns the query results. Otherwise, it will check its model repository to see if the model already exists. If the model exists, the QDS agent launches the scenario agent, which will in turn negotiate with the models, run the models, and have them return a copy of the simulation input and output to the QDS agent and send another copy to the database agent to be stored. The QDS agent will then return the query results to the user. In case the model is not found in the model repository and the user indicates that he/she is interested in creating a new model, the design agent will be launched. After the user finishes, the QDS agent will compile the new model and then launch the scenario agent to let it do its job. If the user wishes to further analyze the query results, the QDS agent will forward the results to a data analysis agent. The purpose of having the QDS agent is to assist the user to make the best use of the system and reduce unnecessary manual interventions.

6. Conclusions and Future Work

Web–based federated simulation is a young discipline and an area of active research [Miller et. al, 2001]. The central issues in this area include reuse of existing simulation models to quickly and cheaply develop more complex simulation systems and the interoperability between disparate simulation systems on the Internet to allow collaboration among independently crafted systems to solve complex simulation challenges. In this paper, we have introduced Jini and XML as two key technologies that will have significant impact on the development of Web–Based federated simulation systems. We have envisioned and sketched out an initial design of a fully distributed and highly extensible Web–based federated simulation system. We have also presented the design and implementation of a simulation system that is oriented toward our vision.

To get to the point of actually delivering a system as we have envisioned, work needs to be done in several areas.

- Work out a common DTD or a standard meta–data interchange format for the simulation domain.
- Develop common interfaces for importing and exporting simulation data.
- Fully implement and deploy the federated simulation system in a distributed environment.

7. References

[Berners–Lee, 2000] Berners–Lee, T. XML and the Web. (September 6, 2000). http://www.w3.org/2000/Talks/0906-xmlweb-tbl/slide1-6.html

[Bosak and Bray, 1999] Bosak, J. and Bray T. XML and the Second–Generation Web. (1999). http://www.sciam.com/1999/0599issue/0599bosak.html

[Cover, 2000] Cover, R. XML Messaging (IETF). (July 2, 2000). http://www.oasis-open.org/cover/xmlMessagingIETF.html

[Ge, 1998] Ge, Y. (1998). Development of a Web–Based Simulation Environment Using Java Bean. Masters Thesis. The University of Georgia.

[Jini, 2000] Jini Connection & JavaSpaces Technologies. (2000). http://developer.java.sun.com/developer/technicalArticles/jini/

[Jini Architecture Specification, 2000] Jini Architecture Specification. (2000). http://www.sun.com/jini/specs/jini1_1.pdf

[Jini Core, 2000] Jini Technology Core Platform Specification. (2000). http://www.sun.com/jini/specs/core1_1.pdf

[Kleinman, 2000] Kleinman, R. Jini and Enterprise JavaBeans Technologies: the Distributed Client Meets the Distributed Service. http://developer.java.sun.com/developer/technicalArticles/jini/espressoman/

[Kuhl et al., 1999] Kuhl, F. Weatherly, R. Dahmann, J. (1999) Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall PTR.

[Laurent and Biggar, 1999] Laurent, S. and Biggar, R. (1999). Inside XML DTDs. McGraw–Hill.

[Miller et al., 1997] Miller, J. Nair, R. Zhang, Z. and Zhao, H. (1997). JSIM: A Java–Based Simulation and Animation Environment. *Proceedings of the 30th Annual Simulation Symposium.* Atlanta, GA. pp. 31–42.

[Miller et al, 1998] 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*. Washington, DC, pp. 373–381.

[Miller et al., 2000] Miller, J. Seila, A.F. and Tao, J. (2000). "Finding a Substrate for Federated Components on the Web". *Proceedings of the 2000 Winter Simulation Conference (WSC'00)*, Orlando, Florida (December 2000).

[Miller et al., 2001] Miller, J. Fishwick, P. Taylor, S. Benjamin, P. and Szymanski, B. Research and Commercial Opportunities in Web–Based Simulation. *Simulation Practice and Theory (SPT).* Special Issue on Web–Based Simulation, Vol. (2001) pp. Elsevier Science. (to appear)

[Nair et al., 1996] Nair, R. Miller, J. Zhang, Z. (1996). A Java–Based Query Driven Simulation Environment. *Proceedings of the 1996 Winter Simulation Conference. Coronado, California*. pp. 786–793.

[Nair, 1997] Nair, R. (1997). JSIM: A Java–Based Query Driven Simulation and Animation Environment. Masters Thesis, The University of Georgia.

[RDF] http://www.w3.org/RDF

[Roth, 2000] Roth, B. (2000). An Introduction to Enterprise Java Beans Technology. http://developer.java.sun.com/developer/technicalArticles/Beans/IntroEJB/index.html

[Seila et al., 1999] Seila, A.F. and Miller, J.A. (1999). Scenario Management in Web–Based Simulation. *Proceedings of the 1999 Winter Simulation Conference (WSC'99)*. Phoenix, Arizona. (December 1999) pp. 1430–1437.

[Tao, 2000] Tao, J. (2000). HLA–Compliant Distributed JSIM. Masters Thesis. The University of Georgia.

[Venners, 1999] Venners, B. The Jini Vision. Java World. (August 1999). http://developer.java.sun.com/developer/technicalArticles/jini/JiniVision/jiniology.html

[W3C XML Site] http://www.w3.org/XML

[Xiang, 1999] Xiang, X. (1999). Use of Agents to Control the Execution of Simulation Components. Masters Thesis. The University of Georgia.

[XLL] http://www.w3.org/XML/Linking

[XML, 1998] Extensible Markup Language (XML) 1.0. (W3C Recommendation 10–February–1998). http://www.w3.org/TR/1998/REC-xml-19980210

[XML Protocol] http://www.w3.org/2000/xp

[XML Protocol Comparisons] XML Protocol Comparisons. (2000). http://www.w3.org/2000/03/29-XML-protocol-matrix

[XML Schema] http://www.w3.org/XML/Schema

[XML Signature] http://www.w3.org/Signature

[XSL] http://www.w3.org/Style/XSL

[XSLT] http://www.w3.org/TR/xslt

[Zhang, 1997] Zhang, Z. (1997). A Java–Based Simulation and Animation Environment: JSIM's Foundation Library. Masters Thesis. The University of Georgia.

[Zhao, 1997] Zhao, H. (1997). A Graphical Designer for JSIM. Masters Thesis. The University of Georgia.