Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

INTEGRATING DISTRIBUTED SIMULATION OBJECTS

Joseph A. Heim

Industrial Engineering, 352650
University of Washington
Seattle, WA 98195

ABSTRACT

Creating comprehensive simulation models can be expensive and time consuming. This paper discusses our efforts to develop a general methodology that will allow users to quickly and efficiently create high fidelity simulation models by linking independent model objects distributed across the Internet or enterprise intranets. The result of linking these models is a *model network* that can be used to evaluate the aggregate performance of the *system* as well as investigate the interactions and performance of the individual *component models*. Our approach for creating a plug-and-play model integration environment is based on the principles of object-oriented programming and distributed object computing. Drawing on advances in language and network communication technology, we continue to refine an early proof-of-concept prototype called ***ENVISION*** (ENVironment for Integrating Simulation models Interactively Over Networks). The primary objective is to create a testbed system that will help us better understand how manufacturers might actually use this type of modeling facility if it was available.

1 MODEL INTEGRATION

For manufacturing firms, the consequences of broadened competition and technological advance are succinctly captured in two metrics: *time*, the profound pressure to respond synchronously to the rapid ebb and flow of the market; and *complexity*, the expanded scope and functionality of products, the increasing number and dispersal of participants, the new technologies and production processes to be adopted, and the distributed nature of the entire product realization process. Decisions become less intuitive as the complexity of the systems increase, the time to make good decisions is shortened and the cost to recover from incorrect decisions can be substantial. Understanding the criteria that define the performance of the individual components is difficult; anticipating the aggregate performance of these new manufacturing systems when the markets, partners, products, technologies and processes continue to change presents enormous risk and challenge.

Simulation models have the potential to provide much of the information needed to reduce the risks associated with the design, analysis and operation of complex systems for product realization. But overarching, integrated models, with the scope to adequately reflect the interdependent performance of the system components, are expensive and difficult to construct; in most instances we build these models from scratch. We need a means of creating a rich depiction of systems from reusable models rather than starting anew each time. Model integration therefore has two primary objectives: provide a comprehensive understanding of the overall *system* as well as its component parts; and, leverage the intellectual resource investments that models represent.

Model integration is used to join individually and independently functioning models of the component parts to create comprehensive model networks that can share data and coordinate their modeling activities. The individual models, such as work cells, machine tools, or material handling subsystems, represent portions of the larger system aggregate. The individual models can also be programs that evaluate system performance, such as a capacity planning model comparing the performance of alternative production configurations represented by independent models. Different system designs can be created by modifying the parameters of participant models or by changing model elements representing manufacturing system components—unplug one model and plug-in another.

By allowing the model builder to use the knowledge abstracted in existing models, model integration reduces programming effort, simplifies model validation and increases the breadth of design options that can be considered in the time available. Model integration is also an effective strategy for development of large models, since interoperability supports the selection of

the language and environment most appropriate to each element of the design effort. For instance, we could have one or more models that were computationally intensive and others that would run adequately on a small workstation or personal computer. With distributed model integration models remain on the computer providing the appropriate resources; their integration is a matter of network data communications.

In this paper we present a methodology for integrating distributed simulation models that is vendor, language- and platform neutral. The fundamental concepts used to create comprehensive system models from distributed models is based on object-oriented programming, client/server relationships and intelligent agent assisted model network construction.

2 COMPONENT-BASED MODELING

Much of the cost associated with modeling can be attributed to the approach we take towards model construction and software development in general (Cox 1996). In the early 1800s, classical manufacturing was in a similar situation: products were made by skilled craftsmen, each component fashioned in a cut-to-fit fashion, so product costs were high. Interchangeable component parts ushered in the industrial revolution. Because skilled workmen were no longer required for product assembly, the cost of manufactured products were significantly lower, and, accordingly, many more people were able to afford manufactured products.

Model building is still in the craftsman-mode of production. It needs to move from the classical, cut-to-fit handiwork approach to an industrial-based method of fabrication from interchangeable component parts. Model "assemblers" should be able to integrate the model component parts in a plug-in manner, thus minimizing the time, cost and expertise required to construct comprehensive models within the context of their organization. The basic methodology we want to create reminds us of the Lego-blocks children use to build complex representations of machines and buildings, and the Plug-and-Play technology that allows us to change personal computer hardware components without coordinating complicated switch settings and software drivers.

2.1 Plug-and-Play Modeling

Although we have to be careful to not push it too far, building our own stereo systems—the way we select and assemble the components we need to play and record music—can be a somewhat more accessible metaphor for explaining how one might construct plug-and-play models of complex manufacturing systems.

By adhering to the guidelines for proper mating of physical connectors and matching appropriate electrical characteristics, a variety of stereo components can be combined to create a wide range of audio systems: a minimal configuration providing adequate presentation of the broadcast signal or recorded media; at the opposite end of the spectrum, we can use audio components of greater fidelity to present a sound that is very rich and difficult to distinguish from a live performance. We can easily add new components to the system, substituting higher fidelity components where we feel we gain the most benefit (e.g., volume versus quality of sound appears highly correlated to age and income of the owner), and replace aggregated subsystems (combined preamplifier, amplifier and tuner) with a set of components that individually accomplish the same functions with greater control and precision (e.g., ability to adjust frequency curves).

The basic result of each configuration is the same, the ability to hear broadcast or recorded music. The difference is in the precision, quality and fidelity of reproduction, a result of the particular component parts we used and their interactions. Interoperability of the components is based on well defined protocols for communication among the components. For stereo systems the protocols are a set of standards for physical and electrical interconnection of audio components.

2.2 New Integration Technology

The approach we propose is obviously then not a unique concept but rather a synthesis and extension of many ideas and developments published (Fishwick 1996; Zeigler 1990) as well as many of the modeling researchers with whom we have discussed most of the issues presented in this paper. (Delen, Pratt and Kamath 1996; Kamath, Pratt and Mize 1995)

The technologies available now and those on the horizon anticipate important applications in many areas beyond simulation and other forms of modeling activities. (Hawker and Waskiewicz 1996) But for simulation and modeling, advances in network communications and programming languages provide opportunities to create the standards and protocols that will allow simulation models to interoperate in the much the same way that physical and electrical protocols allow audio components to be interconnected and function. (National Research Council 1994, 1995)

The most important network communication and user interface developments are HTTP (HyperText Transfer Protocol) and web browsers (e.g., Netscape's Navigator and Microsoft's Explorer). Java, a network-centric, object programming language provides a platform-independent way of creating model objects,

and CORBA (Common Object Request Broker Architecture) provides an open object infrastructure, or middleware, that allows simulation objects created with object-oriented as well as procedural programming languages to interoperate across the data communications networks. (Vinoski 1997; Gosling and McGilton 1996; Nair, Miller and Zhang 1996) These technologies provide a vendor- and language-neutral foundation on which model builders anywhere in the world can construct comprehensive systems models by composition and synthesis of component models (software or model objects) accessible on the Internet.

The integration of distributed simulation models is based on three fundamental themes: (1) models are *objects*; (2) they communicate with one another in *client/server* relationships by passing messages; and (3) each model is represented by an *agent* that explains the capabilities of the model and assists with integration of that model. A few words about objects, client server relationships and agents will clarify our approach to distributed model integration.

2.3 Programs and Objects

An enormous object-oriented programming and design literature is available, so only a brief overview of objects and object-oriented simulation is presented here. (Buss 1996; Joines and Roberts 1994; Dahl and Nygaard 1966) Three basic principles define object oriented programming: *encapsulation*, the way objects hide implementation and associated data but advertise functionality; *message passing*, a strict protocol by which objects communicate and request performance of advertised functionality by other objects; and *classes and* inheritance, a means of organizing the kinds objects you create to maximize code reuse and minimize maintenance efforts. (Goldberg and Robson, 1983) A significant benefit of object-oriented programming is the reduction in *cognitive distance* between the world you wish to represent in the computer and the mechanisms you have available to accomplish that representation. Object oriented programming does this by preserving the decomposition of the system in the computer code you create. (Booch 1986)

2.4 Objects as Simulation Models

For instance, if you were creating a model of a manufacturing plant you might want to represent the machines, routings, work in process, the tools and fixtures, the customer orders and the workers. Using an objected oriented approach, we identify the “things” or entities in the system and the relationships among the entities--what they do to accomplish the objectives of the system. Our application entities or *objects* might be the customer orders, workers, routings, machines, tools and fixtures, and work in process. Each of these *objects* would be represented by a coherent chunk of code that contained all *functionality* for that object as well as the *state* of the object. The functionality would be that set of activities the object would perform if asked, and the state of the object would be the value of all variables describing that particular object. For example, the customer order object would be able to answer questions about its internal state such as “what kind of product are you,” or “what is your due date.” The machine objects might respond to messages such as “begin busy state with this order.” The result of that message would be that the machine object receiving the message would change the value of the internal state variable representing its busy/idle status.

2.5 Client/Server Relationships

The most widely dispersed example of client/server relationships is the world wide web (WWW). Our *browsers* (Netscape and Internet Explorer) are the clients and the applications providing information and data are the servers. We say that the *requests* come from *clients* and that the *server responds* to those requests. In more advanced applications the client or server attribution is likely to be dynamic, based on the context of the communicating program processes: sometimes a program process will be a server, but in other contexts it may also be a client of another program process.

For instance, its easy to envision a situation in which our WIP order objects, machine objects and material handling objects could be both clients and servers. An order (client) requests that a machine object (server) perform some transformation; the machine object (client) in turn requests that the material handling system (server) transport the WIP order from its present location. In all cases, servers are not concerned with the source of the request (in object terms, the request is a message) except to know where the results of the request must be returned. Of equal importance, the client is not concerned about the manner in which its request is accomplished by the server (the server encapsulates, or hides the manner in which it computationally achieves its activities). This kind of relationship among program processes provides great flexibility for implementation. Since clients have no concern about internal changes to implementation, revisions and improvements to the server-side of the relationship can proceed independently. The server is only responsible for continuing to respond to its

previously advertised capabilities (services) in the agreed upon manner (the protocols for exchange).

This means that we can substitute modeling implementations, even going so far as to move the model to a new platform for higher speed computation, and the users, the remote clients of that model will not have to make changes to the manner in which the models interact. Improvements and maintenance may proceed independently of users of the model services.

2.6 Agents

Agents are software programs that execute specific tasks on behalf of another party. The rules under which they operate can be created and defined in various ways and of varying complexity and sophistication. Agents can also be characterized by whether they are deliberative or reactive, that is do they respond to stimuli or do they have an internal reasoning logic which allows them to participate in "thoughtful" processes. For instance, an email agent would be reactive. It would monitor receipt of messages and based on the database of criteria provided, the agent would filter and dispatch each message into the appropriate folders (immediate response, work related, entertainment, no response necessary) for our later perusal and action. A deliberative agent might try to formulate responses to the received email messages by more closely examining the message content.

For our modeling purposes, the agents will provide reactive services. They will help model builders select the appropriate model from among a menu of models, and they will assist in the configuration and operation of the distributed model networks. An important aspect of model integration is the selection of appropriate models to be linked. The agent provides an open ended structure into which we will later be able to incorporate the necessary information and mechanisms to address the semantic issues of model selection more completely.

Our basis for integrating distributed models then is to create the framework and methodology in which individual models can become message-passing objects that communicate with one another as both clients and servers. Each model has associated with it an agent that describes the capabilities of the model, its constraints and data needs as well as the data it produces and coordination requirements. The agents for the models also generate the interface programming logic needed to participant in the distributed modeling activity.

3 ENVISION: AN ENVIRONMENT FOR MODEL INTEGRATION

Earlier model integration efforts by the author produced an unrefined, but operational, proof-of-concept prototype called ENVISION. The prototype helped us explore many of the architectural issues involved in creating an open systems integration methodology, and demonstrated that linking distributed object-models (analytical as well as simulation) was feasible for tasks that were otherwise quite difficult and time consuming when the models were integrated in more traditional ways (e.g., file translation, hand carrying results between programs, customized interfaces). (Heim 1994)

But the original version of ENVISION required a combination of C, Smalltalk and FORTRAN routines, and too many low-level operating system calls to be of practical application. However, the availability of data communications-enabled, object-based programming languages such as Java (Gosling and McGlinton 1996), and the ability to construct high-performance, platform-independent graphical user interfaces has encouraged us to revisit the implementation of ENVISION. The motivation has come from our recognition that many of the 375,000 smaller manufacturers in the United States would likely benefit from greater access to simulation modeling, but most have not adopted the technology. (National Research Council 1993) Recent investigations, therefore, are concerned with understanding how the firms would use such a modeling facility and from where they would expect to obtain the models for integration.

Rather than reviewing the original implementation of ENVISION in detail, we'll look at the approach we have taken to convert it into a more useful and accessible system. The development effort has been divided into three major tasks:

- Developing a standard methodology to transform individual distributed simulation models into objects with full network communicating abilities;
- Creating agents to assist in selection and integration of models;
- And developing an environment that brings together the tools and mechanisms to construct model networks and monitor and control their interactions.

Our objective is to create a testbed system that will help us better understand how manufacturers might actually use this type of modeling facility if it was available.

3.1 Model Objectification

The first research task is to develop a standard manner for describing the model interfaces that will allow independent models to interoperate as an association of distributed model-objects. Although all new code will be written in Java for reasons of portability and ease of extension, we are not likely to have models of any consequence written in Java. Models created with "legacy" simulation languages will continue to be the major source of potential model objects, so we create an object shell, or wrapper, that surrounds the model and enables it to send and receive messages in a standard object-like manner. This provides the "outer" communications edge of the wrapper. The other portion of the wrapper, the "inner" edge of the communications standard is concerned with creating a mapping facility that translates the language of the host into the standard at the opposite edge of the object wrapper. For instance, some models read and write to files or particular hardware ports on the computer. These must be mapped into message-format leaving the model and converted from message format into file format when coming into the model from outside its wrapper.

The wrapper or shell provides the mechanisms for translating a model into a standard format. But that doesn't get the model object connected to other model objects, possibly on other computers on the Internet. A concept called *middleware* provides the network interoperability among the distributed model objects.

Middleware is the translating layer between heterogeneous clients and servers (both of them are objects); those objects could be on different computer platforms, they could be written in different languages, and they could be located anywhere on the Internet or an enterprise intranet. Because we intend to develop an open systems architecture and methodology for model integration, we will avoid vendor-specific implementations that are not open to current and developing standards. The most open middleware choice at this time is CORBA (Common Object Request Broker Architecture), an industry specifications standard for distributed computing. (Vinoski 1997)

Object Request Brokers (ORBs) are the software analog of the hardware backplane, or bus, in which electrically standardized pin assignments and physical configuration allow a set of electronic subassemblies (e.g., printed circuit boards) to be combined to accomplish some system objective. Following this analogy, our collection of encapsulated simulation models are our printed circuit boards and the ORB is our backplane for connecting our model-objects. The set of messages to which our models will respond are the electrical standards for pin assignment and the ORB assures that pin assignments among the model-objects are matched correctly.

Model objects require one more layer of software to allow them to interoperate without concern for platform, language or location. This layer of software is needed to create the an interface that crosses the Internet. An interface generator, under the control of the model agent, creates the appropriate IDL (Interface Definition Language) with which the encapsulated models will communicate. The IDL allows the creators of individual component models (such as a machine model or material handling system model) to specify in a standard manner the functions their models provide for access by potential client models without concern for what kind of computer will interoperate or in what language the remote model participants were written.

3.2 Building Model Agents

The purpose of model agents is to facilitate the construction of network models. Model agents know what their model object can accomplish, what data they need to perform those actions and what information the model will provide as it executes. For example, when a person wants to construct a network model, they download the model agents representing all of the model objects needed. The agents will configure the interface of their respective model objects and provide the information necessary to configure the network model. The agents also help the network model builder select the appropriate model objects from those available on the network by providing semantic information about the model objects they manage. The builder of the individual models creates each model agent using tools and templates also developed in this project.

The agent is created and maintained separately from the model. For instance, there could be several implementations of the same modeling function and all could be represented by the same agent. The agent would help the user select the most appropriate implementation based on execution speed, size of the task to model, ancillary capabilities (e.g., graphical or animation output).

As the specificity, functionality and intellectual property content of models from equipment suppliers and other commercial sources grows, the importance of retaining control and restricting access will become important. (Cox 1996) Agents can provide the intellectual property controls and accounting mechanisms needed to allow customers considering adoption of their equipment access to their high fidelity models. Vendors could charge users for access to their models and rebate those costs if the equipment modeled was subsequently purchased from them.

3.3 An Environment for Constructing Model Networks

The final task is the definition and construction of an integrated environment in which model objects can be accessed and connected into network models. The network model construction environment must have a means of determining the availability of models, link the models into an appropriate federation and then provide the mechanisms for controlling the execution and capturing the results of the modeling activities. We are using a browser-based GUI to construct and operate the model network. LDAP (Lightweight Directory Access Protocol) defines a reasonably simple mechanism for Internet clients to query and manage an arbitrary database of hierarchical attribute/value pairs over a TCP/IP connection. An implementation of LDAP client-server protocol will be used to retrieve and manage a directory for locating available model objects.

When distributed models function in a model network their progress and activity must be coordinated, that is, results from one or more models may need to be provided as input data for several other models; interactions and dependencies among components of the manufacturing systems are not limited to simple pair-wise associations. The model construction environment is also responsible for defining modeling coordination needs and then accomplishing those coordination functions with only minimal input from the people building the model networks. The number of decisions to be made by the model builder should be minimized as well as the responsibility for identifying when data exchanges occur among participant models. Coordination of the models remains an unresolved issue, although we believe that the work by Zeigler (1990) and others on parallel/distributed model execution can provide important guidelines on developing the necessary coordination mechanisms for the distributed simulation objects. (Chow and Zeigler 1994)

4 SUMMARY

Modeling reduces time, cost and risk while producing information needed for design, analysis and operation of complex production systems. However, the costs of modeling, the expertise required, and the need to create models that are context representative are impediments to more wide spread adoption of simulation technology. We need a means of easily creating comprehensive system representations from reusable models rather than starting anew each time. In this paper we presented a methodology for integrating distributed simulation models that is vendor-, language- and platform-neutral. The fundamental concepts are based on object-oriented programming, client/server relationships and intelligent, agent-assisted model network construction. A primary objective of model integration is to obtain a better understanding of the system, the performance of the component parts and their interactions. We want to be able to construct models that become good information systems and expose errors and stimulate their correction before the systems are operational. (Brooks 1995)

REFERENCES

Booch, G. 1986. Object-Oriented Development. *IEEE Trans. Software Engineering*, Vol. SE-12, No. 2, February, 211-221.

Brooks, Fred. 1995. *The Mythical Man-Month*, Anniversary Edition, Addison-Wesley, Reading MA.

Buss, A.H. and K.A. Stork. 1996. Discrete Event Simulation on the World Wide Web Using Java. *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, 780-785.

Chow,A.C. and B.P. Zeigler. 1994. Parallel DEVS: A Parallel, Hierarchical Modular Modeling Formalism. *Proceedings of the 1994 Winter Simulation Conference. Lake Buena Vista*, FL, 716-722.

Cox, Brad. 1996. *Superdistribution: Objects as property on the electronic frontier*. Addison-Wesley, Reading, MA.

Dahl, O.J. and K. Nygaard. 1966. SIMULA-An ALGOL-based Simulation Language. *Communi-cations of the ACM*, Vol. 106, September, 671-678.

Delen, D., D. Pratt and M. Kamath. 1996. A New Paradigm for Manufacturing Enterprise Modeling: Reusable, Multi-Tool Modeling. *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, 985-992.

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

Goldberg, Adele and David Robson. 1983. *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, Reading, MA.

Gosling, J. And H. McGilton. 1996. The Java Language Environment: A White Paper. http://www.JavaSoft.com/doc/language_environment/

Hawker, Scott and Fred Waskiewicz. 1996. Agility enabled by the SEMATECH CIM Framework. In *SPIE Proceedings, Plug and Play Software for Agile Manufacturing*, Boston, MA, 69-95.

Heim, J.A. 1994. Integrating Distributed Models: The Architecture of ENVISION. *International Journal of*

Computer Integrated Manufacturing, Vol. 7, No. 1, 47-60.

Joines, J.A. and S.D. Roberts. 1994. Design of Object-Oriented Simulation in C++. *Proceedings of the 1994 Winter Simulation Conference. Lake Buena Vista*, FL.

Kamath, M., D.B. Pratt and J.H. Mize. 1995. A Comprehensive Modeling and Analysis Environment for Manufacturing Systems. *Proceedings of the 4th Industrial Engineering Research Conference*, 759-768.

Koonce, D. , R. Judd and C. M. Parks. 1996. Manufacturing systems engineering and design: an intelligent, multi-model, integration architecture. *International Journal of Computer Integrated Manufacturing*. Vol.9, No.6., 443-453.

Nadoli, G. and J.E. Biegel. 1993. Intelligent Manufacturing-Simulation Agents Tool (IMSAT). *ACM Trans. on Modeling and Computer Simulation*, Vol. 3, No. 1, 42-65.

Nair, R.S., J.A. Miller and Z. Zhang. 1996. Java-Based Query Driven Simulation Environment. *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, 786, 793.

National Research Council. 1993. Manufacturing Studies Board. *Learning to Change: Opportunities to Improve the Performance of Smaller Manufacturers.*

National Research Council. 1994. Computer Science and Telecommunications Board. The Open Data Network: Realizing the Vision of an Integrated National Information Infrastructure. In *Realizing the Information Future: The Internet and Beyond*, 43-111.

National Research Council. 1995. Computer Science and Telecommunications and Manufacturing Studies Boards. *Information Technology for Manufacturing: A Research Agenda*.

Nwana, H. S. 1996. Software Agents: An Overview. *The Knowledge Engineering Review*, Vol. 11, No. 3, 1-40.

Vinoski, S. 1997. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, Vol. 14, No. 2, February.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models*, Academic Press, San Diego, CA.

AUTHOR BIOGRAPHY

JOSEPH A. HEIM is an assistant professor of Industrial Engineering at the University of Washington in Seattle, Washington. His teaching responsibilities include computer integrated systems, simulation, inventory management, and scheduling. Heim's research focuses on the use of distributed discrete event simulation models to coordinate complex collaborative tasks. Professor Heim is a member of ASEE, ACM, IIE and SME.