Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

SIMULATION SOFTWARE COMPONENT ARCHITECTURE FOR SIMULATION-BASED ENTERPRISE APPLICATIONS

Charles R. Harrell

Department of Manufacturing Engineering
Brigham Young University
Provo, Utah 84602, U.S.A.

Donald A. Hicks

Senior Vice President
PROMODEL Corporation
Orem, Utah 84058, U.S.A.

ABSTACT

This paper examines trends and technologies leading towards simulation-based enterprise applications. Component, internet and distributed computing technologies are presented as enablers of simulation-based enterprise applications. Examples are given of typical applications that can take advantage of distributed simulation components. The goal of this paper is to present a high level component architecture that will work in current enterprise information technology (IT) environments.

1 INTRODUCTION

Historically, discrete-event simulation has been viewed as a standalone, project based technology. Simulation models were built to support an analysis project, to predict the performance of complex systems, and to select the best alternative from a few, well defined alternatives. Typically these projects were time consuming and expensive, and relied heavily on the expertise of a simulation analyst or consultant. The models produced were generally "single use" models which were discarded after the project.

Over the last few years, the simulation industry has seen increasing interest in extending the useful life of simulation models by using them on an ongoing basis (Hicks 1998). Front-end spreadsheets and push-button user interfaces are further making these models accessible to decision makers. In these flexible simulation models, controlled changes can be made to the model configuration and management rules for decision support throughout the system life cycle. This trend is growing to include dynamic links to databases and other data sources enabling entire models to be actually built and run in the background using data already available from other enterprise applications.

The trend to integrate simulation as an embedded component in enterprise applications is part of a larger trend to develop software components that can be distributed over the internet. This trend is being fueled by three new information technologies (Wolf, Smith, and Chi 1998): (1) component technology which delivers true object orientation, (2) The internet or world wide web (WWW) which connects business communities and industries, and (3) distributed computing standards such as Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM).

The objective of this paper is to outline a generic software component architecture that can be used to embed simulation computational power into other applications producing simulation-based enterprise applications. We intend to first outline the problem environment and resulting software design requirements. Then we will discuss the functional requirements that enterprise computing environments impose upon simulation software design. We will outline a high level component design that meets these requirements, discuss the technological and software development issues that arise from the implementation of this design, and finally draw conclusions about the implications this architecture has for embedding simulation technology into the enterprise computing environment.

2 PROMISE VERSUS PRACTICE

It is difficult to argue with the economics of reusing technology that has already been paid for to get more return on the initial investment. If Company X pays for a simulation project it is often paying largely for the answers that the project will get them. However, if the simulation model that yielded those answers can be used for a second or third project, Company X can save considerably by getting the benefits of the projects without having to pay each time for a new model build.

While this is sound economics in theory, in practice companies often find it extremely difficult to reuse simulation models. Discrete-event simulation is frequently used to examine and analyze very detailed problems, and hence produces very detailed simulation models. Rarely are the details of two operational problems the same, even

if the problems are dealing with similar issues. This means that, from a detailed modeling perspective, every situation is a unique situation, and cannot be modeled generically with a reusable simulation.

3 THEORY OF PROBLEM DOMAINS

One of the more difficult tasks in designing and building discrete-event simulation models is determining the appropriate level of detail. The modeler must translate the real system into an abstract system of objects, object interactions with other objects, and object interactions with the system environment. A modeler who seeks to create an abstract representation of some operation is faced with three options: (1) model the operation in detail to capture actual cause and effect relationships, (2) model the operation without much detail to capture only some of the interactions, or (3) "black box" the operation expressing its presence through a math model or probability distribution.

A model builder will make his "level of detail" decision based on whether he thinks that additional detail will significantly affect the output statistics he is attempting to calculate. Thus, the modeler may decide that the operation in question is not relevant to the question he is interested in answering, and choose to simply ignore its effects.

The ability to resolve "level of detail" decisions based on their relationship with overall objectives is not a trivial skill. Experts make these decisions on a case by case basis, and rely on extremely complicated and difficult to define heuristic decision making procedures. Ask a consultant how he knew to black box one issue and collect data on another, and you're likely to get an answer such as, "Because I've done a number of these kinds of problems before," or "Based on experience."

The important concept here is that there seems to be general classes of problems that are commonly simulated, and that behave similarly in terms of their structure and guidelines for level of detail selection. For example, call center operations are simulated quite often to answer one of several questions. Every single call center is different, so if you wanted to model each call center precisely to determine some statistic to which the smallest details are relevant (such as the number of sticky notes used by the staff each day), you could expect that you would have to model each one from scratch. Fortunately, most call center simulations are focused on determining several key metrics such as staff utilization, customer service, and customer waiting time parameters.

For this class of problems, which we term a "simulation problem domain," the simulation model required to get the answers most users want is consistent and definable. An experienced simulation developer should be able to define all of the data requirements needed to build the call center model, and then further be able to construct appropriate models from those data tables. Hence, the developer can write software that would, for the call center problem domain, consistently generate simulation statistics given only the call center data model.

The concept of simulation problem domains is quite important. It establishes a new framework for evaluating the use of simulation technology in decision making, and breaks the idea of simulation modeling into two broad application types: simulation projects, and embedded simulation-based decision support tools.

Some sample simulation problem domains that we've identified to date include:

- Call Center Applications
- Emergency Room Modeling
- Manufacturing Capacity Planning
- Inventory (Demand-Supply) Simulation
- Business Process Simulation (Harrell and Field 1996)

4 SOFTWARE COMPONENTS AND PROGRAM SERVICES

The requirements placed upon a software program that facilitates embedded simulation applications are quite different from the requirements of a program that supports user driven model development. In traditional simulation model building, the simulation software application acts as a model development environment. It is used on a single workstation or PC and data is entered primarily by the user. The simulation development environment is a general modeling environment that must support a nearly infinite variety of modeling situations. Hence, the modeling language and the software will be used in ways that the designer never anticipated, making it likely that the software will throw exceptions on a regular basis. It is this core issue, designing software to handle infinite use cases, that makes simulation software development environments extremely difficult to make robust and error free.

In simulation development environments, a graphical user interface is the primary means of interacting with the application. The application is nearly always run as a stand-alone executable file, and will occasionally pull data from other sources, acting as a client.

Simulation-based applications, on the other hand, will not involve direct simulation by humans. Instead, the simulation application takes data from existing sources, applies discrete-event simulation to the problem as defined by the simulation problem domain, and creates new data output which is put back into the database. The simulation-based application effectively maps input data tables to output data tables in the same enterprise database.

The simulation-based application is likely to adopt a distributed architecture. The user who takes an action that triggers a requirement for simulation may not even know

he's asking for a simulation to be run. Rather, the user is using some application on the network, and the application is the one requesting the simulation. The simulation program in this case is required to be a server, rather than a client.

Additionally, the simulation software program must have an extremely well defined and documented programmatic interface and extensive APIs. The program must operate as a software component that is designed to support requests for its services. In today's enterprise computing environment, this means adopting a published and widely accepted interface standard such as CORBA or DCOM.

Table 1 shows a comparison of the application characteristics found in simulation projects versus embedded simulation applications:

Table 1: Requirements for Simulation Projects Versus Embedded Simulation Applications

Characteristic	**Simulation Projects**	**Embedded Simulation**
Nature of problem	Strategic/Planning	Tactical/Operational
Application	General	Domain Specific
Interactive Role	Client	Server
Primary interface	Graphical (GUI)	Programmatic (CORBA, DCOM)
Sources of input data	User, flat files	ERP, Corp. database
Model Builder	Human	Software routine
Destination of output data	Flat file report	ERP, Corp. database
User of Output	Human	Other software objects

To summarize, a simulation model development environment is a robust, graphical-oriented application that generally runs as a stand-alone application and solves any possible problem equally well. A simulation-based application requires a simulation component that can act as a server on a local area network (LAN) or wide area network (WAN), can simulate a few types of problems extremely well and with no exceptions.

5 SIMULATION AND OPTIMIZATION COMPONENTS

Figure 1 illustrates how traditional simulation development environments are typically constructed. Of course, every simulation software package has its own unique variation to this approach.

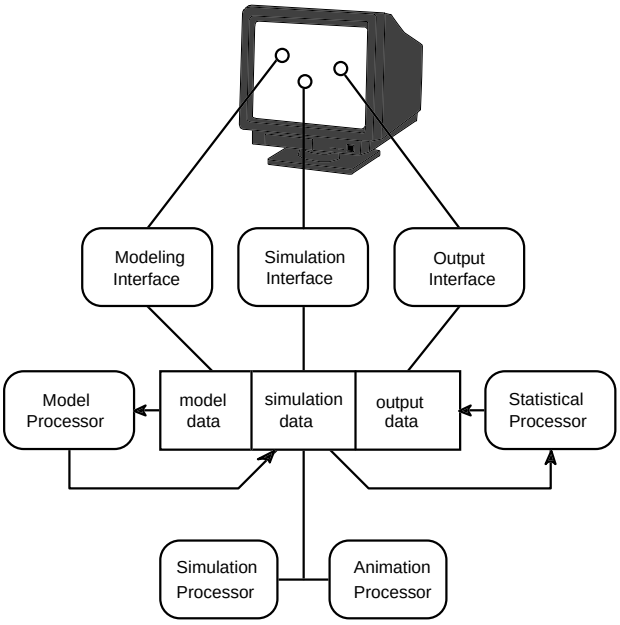

Figure 1: Traditional Components of Simulation Software (Harrell and Tumay 1994)

In traditional simulation software architectures the basic software components are all integrated into a single monolithic application. Interfaces between the components are internally and are not exposed to the outside world. This is fine for PC based off-line planning and improvement projects. All of the processing happens on one machine, which optimizes the program for quick response to the user's actions. There is no waiting time for a server and no load placed on the network. However, this is totally unsuitable for integrated simulation applications that don't have any graphical components, and that interact with a large number of external applications written by other developers who probably don't know what interfaces the simulation program has implemented.

It is common in simulation, even for traditional simulation model development, for other external applications to be used. Almost every simulation project uses a spreadsheet to manipulate tabular data, a curve fitting component to convert data into probability models, an optimizer or scenario manager to run experiments, an output report generator, and probably some kind of commercial editor (such as MS Word, PowerPoint, or an HTML editor). These other components should not be overlooked in formulating an architecture for simulation because they are just as important to the process of improving corporate decision making. At the same time, instead of communicating at the API level, these components often use the user himself as the vehicle for transferring data.

Table 2: Software Components Required for Simulation Based Applications

Software Component	Services Provided by Component
Curve Fitting	Translates raw data lists into probability distributions
Enterprise Database	Provides a repository for process definition and transaction data
Simulation Database	Provides a repository for simulation problem specific data
Data Retriever	Retrieves the data required by the simulation database from all sources of data in the enterprise
Simulation Engine Component	Translates simulation database into experimental results database
Animation Component	Displays graphical animation of model experiment
Optimization/Goal Seeking Component	Conducts multiple simulation experiments in order to achieve a specific goal

In simulation-based applications, these same components in the simulation software system, both external and internal, come together to transform and generate new data for decisions. By adopting a pure component architecture, however, the components can be reused and tightly linked for a specific problem domain. Below is a table of the software components that are required to configure a simulation-based application, along with the major services those components will provide.

6 OUTPUT AND ANIMATION ISSUES

In this application framework, the simulation engine is basically a translator, or data mapping device. It is used to translate an input database into an output database. While this is not essentially different from what simulation software packages do today, it is important because the focus and main usage has shifted from model building activities to decision-support activities. In the paradigm of simulation-based applications, model building as an activity is performed when the problem is formulated and structured. Hence, the focus shifts from graphical interaction (both in model development and output) to data model interaction.

Of particular interest is the issue of animation. A simulation component that is used to offer its particular services to other applications may not generate animation directly. In stead it may generate detailed output of graphical object movements and status changes of interest in the model. It is then up to other components to determine how to use this information, stored in the experiment output database, to display visualizations.

Animation, seen from the component perspective, is simply another means of displaying output data. It is really no different than other detailed output data produced by a simulation experiment, except in how it is displayed. Thus, the onus of animating and displaying data graphically is placed on the components in the system that is responsible for display.

7 INTEGRATING SIMULATION-BASED APPLICATIONS INTO THE ENTERPRISE ENVIRONMENT

Interoperability has become one of the most important issues for corporate IT departments and software vendors. What was once a wide open “wild west” of computing has quickly converged around several standard protocols that are relevant to integrating simulation components into enterprise applications.

Corporate databases, mainframes, ERP systems such as SAP R/3, BaanERP, Oracle, and PeopleSoft all provide much of the structural information required to produce simulation models. In addition, these applications, which many companies have already implemented, contain much of the detailed transactional data necessary to complete the building of complete simulation models.

Structured Query Language (SQL) is already a standard protocol for communicating with relational databases. Additionally, most of the major enterprise applications used by companies have well defined and relatively complete APIs which can be utilized by the simulation application to gather the data necessary for a problem.

CORBA and DCOM are now accepted industry standards for distributed computing. CORBA was released in 1994 by the Object Management Group and is designed to implement applications across multiple networks, languages and platforms. DCOM is Microsoft’s solution for distributed applications and is based on ActiveX component technology. DCOM is currently only supported by Windows and a few UNIX systems.

8 CONCLUSIONS

This paper presents concepts and technologies for implementing embedded simulation in enterprise applications. These applications support complex decision-making and shared data. Component software development allows simulation software to be constructed in self-contained modules with well defined interfaces that can be easily embedded in other applications. The internet, CORBA and DCOM permit software components to be

distributed and no longer bound to a stand-alone environment.

This new direction in simulation technology holds exciting opportunities for deploying simulation on a much broader scale than the specialized, project-oriented applications of the past. Simulation is rapidly becoming an integral component in enterprise applications providing powerful, simulation-based decision making capability.

REFERENCES

Harrell, C. and K. Field 1996. Integrating Process Mapping and Simulation, In *Winter Simulation Conference Proceedings* ed. J.M. Charnes, D.J. Morrice, D. T Brunner, and J. J. Swain, 1292-1296.

Harrell, C. and K. Tumay 1994. *Simulation Made Easy*, Norcross, Georgia: IIE Press.

Hicks, D. 1998. Simulation Market Forces Can't be Ignored, In *IIE Solutions*, May 1998, 18-19.

Wolf, Philip M., R. L. Simith, and Y. Chi. 1998. WWW, CORBA and Java: New Information Technologies for Industrial Engineering Solutions. In *Solutions '98 Proceedings*. 1-6.