JAVA-BASED QUERY DRIVEN SIMULATION ENVIRONMENT

Rajesh S. Nair
John A. Miller
Zhiwei Zhang

Department of Computer Science
University of Georgia
Athens, Ga 30602-7404, U.S.A.

ABSTRACT

The concept of Web-based simulation can be finally realized using Java. We have used Java to create a powerful simulation modeling library which is based on the process interaction paradigm. Java threads make it easy for us to implement each process (or active entity) as a thread. Our library, JSIM, supports both simulation and animation thus rendering the model developer's job easier. An important component of our approach is that we integrate our Java Simulator with a Database Management System (DBMS). The conceptual basis for this integration is an environment based on Query Driven Simulation, which allows simulation analysts to study the performance of systems simply by querying a database. Simulation inputs and outputs are stored in databases, and simulation models can be launched as a part of query processing. This integration also resolves a security restriction problem which Java imposes. Most browsers (e.g., Netscape Navigator) restrict Java applets from writing into a file which may be necessary to make simulation results available to the user. Our approach bypasses this problem by writing into a database. Databases are much less of a security concern since they already have substantial authorization mechanisms.

1 INTRODUCTION

Simulation has come a long way since its inception. An increasing number of real world concepts are being modeled and studied using simulation. Recently, simulation has been influenced by an increasingly popular phenomenon - the World Wide Web or WWW, as it is known. The popularity of the Web has propelled it into being a medium for all kinds of applications including simulation.

The concept of Web-based simulation can be finally realized using Java. Java is an object-oriented programming language for the Web. With Java, it is possible to have "executables" on the Web. This provides users with a dynamic environment as opposed to a largely static, text-driven environment that was existent prior to Java. Simulation users can finally view results over the Web in a dynamic environment complete with animation, sound and textual information.

Java is an important advance in Web technology, and one that is particularly important to simulation. Previously, interactivity or minimal dynamism on the Web were accomplished using Common Gateway Interface (CGI) scripts, typically coded in Perl or C/C++. Unfortunately, these scripts run on Web servers, thus limiting how dynamic the display on a remote Web browser could be. Java is fundamentally different in that it allows applets downloaded from a Web server to be run by the browser. Therefore, dynamism is only limited by the speed at which Java code can be executed. Since Java aims for universal portability, browsers provide a byte code interpreter to execute code produced by a Java compiler. Consequently, it is not as fast as purely compiled code (e.g., C/C++) but is faster than purely interpreted code (e.g., Perl). However, in the future it is expected that native code compilers will be available for Java to make its speed competitive with C/C++. It should be noted that since Java source code (.java) is compiled to byte code (.class), source code need not be made available to the Web as it must be for purely interpreted languages. Let us summarize by listing some of advantages of using Java to implement simulation models.

1. Simulation models implemented as Java applets can be made widely available. Anyone that you give permission to that has a Java capable Web browser (e.g., Netscape 2.0 or higher, HotJava 1.0 or higher) could potentially run your simulation model on their local machine.

2. The goal of universal portability means that one simply retrieves an applet and runs it. One does

not have to port to a different platform, or even recompile or relink.

3. Java applets run on a browser allowing a higher degree of dynamism.

4. Java has built-in threads making it easier to implement simulations following the process interaction paradigm.

5. Java has built-in support for producing sophisticated animations. Geometric objects can be readily created, moved, color changed and destroyed. In addition, icons/images (.gif, .jpeg) can be similarly used.

6. Some also see Java as a better C++. It is smaller, cleaner, safer and easier to learn than C++. Notably, it does not provide pointers and automates storage management through the use of garbage collection. Thus, many of the most common C++ errors would be avoided.

Java shows great promise for simulation and animation. We have used these strengths of Java to build a comprehensive library, called JSIM, to support both simulation and animation. One potential problem with Java involves making data persistent. For example, in Netscape Navigator 2.0, applets cannot read or write files at all (see http://www.javasoft.com/java.sun.com/sfaq). Our solution to this problem is to have applets read from and write to a database instead. In fact, a Database Management System (DBMS) is an integral part of our simulation environment. The integration of the simulation and database components is done following the notion of Query Driven Simulation (QDS).

In the rest of this paper, we overview the Query Driven Simulation environment, and then cover the three main aspects of the JSIM library: simulation, animation and database access. This is followed by an example simulation model and an example QDS query. Finally, conclusions and future work are presented.

2 QUERY DRIVEN SIMULATION

Query Driven Simulation is based on the tenet that simulation analysts as well as naive users should see a system based upon QDS as a sophisticated information system which will be able to store or generate information about the behavior of systems the user wishes to study (Miller and Weyrich 1989,Miller et al. 1991a,1991b,Miller et al. 1996a, Miller et al. 1996b). This means that the user must be provided with an easy to use environment where he/she may trigger an action using a simple query language.

The QDS environment principally consists of three communicating processes and several data stores (see Figure 1). The processes include a Web browser (e.g. Netscape Navigator or HotJava), a Web server (e.g. Netscape Commerce Server) and a DBMS (e.g. m-SQL). The data stores consist of models, data and meta-data stored in databases or files accessible to the Web server. The Web server will also store the applets used in the QDS environment such as the QDS Applet, the Design Applet and potentially numerous model applets. The JSIM library simplifies the development of model applets. The QDS environment is aimed towards providing a friendly and easy to use interface to the user. A typical session involving a user and the QDS system can be explained as follows. When the user first contacts the Web server (e.g. httpd), they download the main control applet for the environment, namely the QDS Applet. The QDS Applet upon starting up gives the following options to the user.

1. Find Model Applet
2. Run Model Applet
3. Submit SQL Query
4. Save Results to DBMS
5. Update Model Index
6. Submit QDS Query
7. Design Model

All the options, excepting option 7, can be explained by outlining the functions performed by option 6. If the user elects to submit a QDS query, the QDS Applet first treats the query as a normal SQL query and submits it to the database engine. The query is run against the existing database and the results are returned to the QDS Applet. If the results are adequate, then they are presented to the user. Otherwise, the QDS Applet submits a query to the Model Index in the database to determine the relevant model. Once the entry for the model in the Model Index table is found, the QDS Applet extracts the parameters required by the model applet from the query itself. (For option 2, the QDS Applet requests the user for the parameters.) This is called "defining a scenario". The QDS Applet then requests the Web server for the model applet code. The Web server responds by providing the applet code which is then executed with the appropriate parameters. The model applets use the classes provided in the JSIM library (section 3.4). The results generated by the model execution may or may not be stored in the database

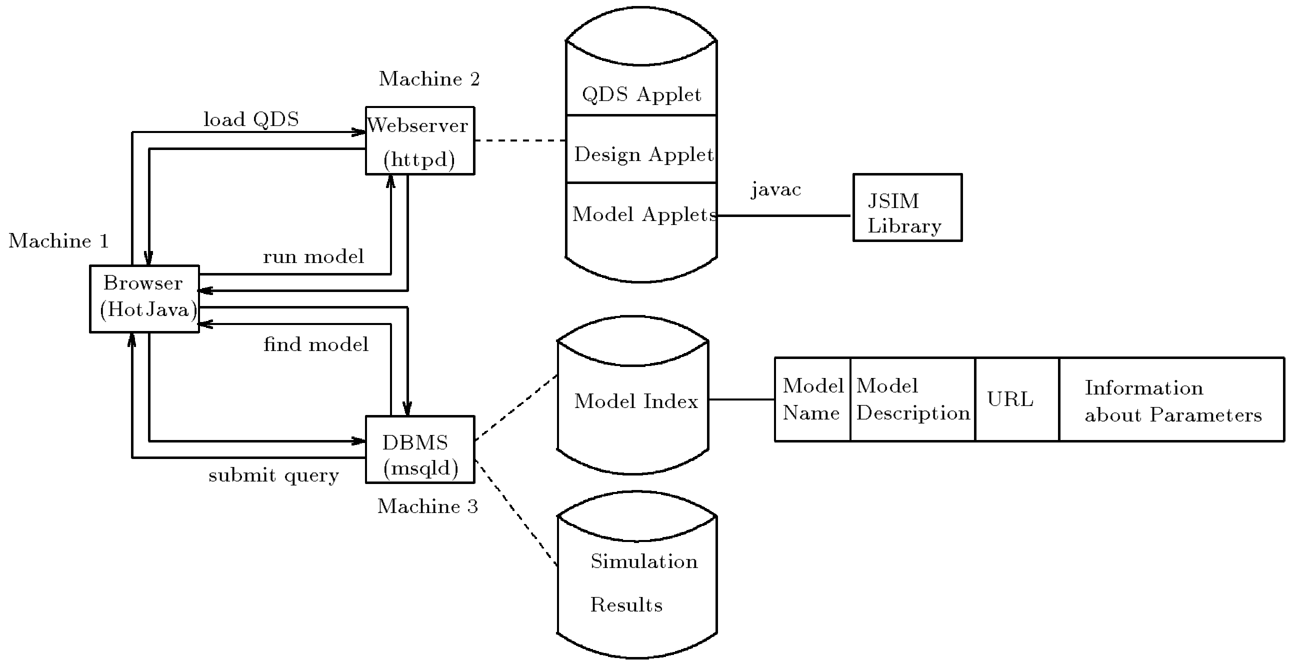

Figure 1 : The QDS Environment

depending on the user's choice. It is also the responsibility of the model applet to display the results of model execution to the user. Option 7, Design Model, is different from the other options in that it is meant for the simulation model builder rather than the user. A special applet called the Design Applet is invoked when the user selects this option. The Design Applet helps the model builder design a simulation model by providing him/her with a graphical interface in which a model can be built primarily by drag and drop operations. Currently, the Design Applet is not fully functional.

3 JSIM:THE JAVA SIMULATION LIBRARY

JSIM, written in Java, is a simulation toolkit with a library of Java classes. JSIM provides most of the features of SIMODULA developed by (Miller and Weyrich 1989). In our implementation, we have built simulation, animation, random variate generation, statistical analysis and database access facilities into the library. In building our library, we followed the process interaction paradigm. Java's built-in support for threads helped us in implementing a true process interaction paradigm. The simulation entities have been coded as threads and are truly independent and may perform actions on their own. Throughout the discussion, we consider a simple simulation example - that of a bank with one or more tellers and multiple customers. The customer arrival time is random as is the service time.

In the following subsection, we discuss our library in greater detail. We begin with the simulation component.

3.1 Simulation Facilities

We use the process interaction paradigm for simulation since we feel it is closer to real world concepts than the event scheduling approach. Java threads make it possible for us to realize the process interaction paradigm in a straightforward manner.

As can be seen from the bank simulation example of section 4, both customers as well as tellers are independent entities. Each simulation entity, namely customer, teller, etc., is coded as a Java thread. Java's support for multi-threaded programming makes it an ideal platform for an implementation of the process interaction paradigm.

Furthermore, we have provided for both virtual-time clock and real-time clock simulation capabilities. The virtual-time clock simulation is ideal for speedy simulation with little or no machine interference in simulation timings. It is particularly helpful if the real world events that are being simulated are long lived. Virtual-time clock simulation uses the so-called "next-event" strategy for advancing simulation time. Here, the clock value jumps to the time at which the next event is due to occur. It is usually implemented with a scheduler and a Future Event List (FEL). The scheduler first schedules an event by inserting it into the FEL depending on its activation time. It then transfers to the event at the front of the FEL by removing it from the FEL and updating the simulation clock to the time associated with that event. Real-time clock simulation, as the name suggests involves using a real world clock (system clock time obtained using the Java function System.currentTimeMillis) for simulation timing. It is well suited for real-time animation where simulation events are represented by corresponding animation events. In real-time clock simulation, the clock does not move in jumps but progresses smoothly. Entities are delayed by using sleep or suspend which are methods within the thread class.

We also provide for animation of virtual-time clock simulation as is explained in the next subsection.

3.2 Animation Facilities

As mentioned above, we provide for animation of both real-time clock and virtual-time clock simulations. To support real-time clock simulation, we built in animation capabilities into each of the simulation entities. Each simulation entity has its own *draw* method. We used a technique called double buffering to implement multi-thread animation capabilities. The double buffering technique is generally used to reduce flicker during animation. However, we have used this technique to our advantage in a totally different manner. A single *display thread* periodically calls every active simulation entity to draw itself into an offscreen buffer. After the offscreen buffer has been completely updated, the display thread repaints the screen by copying the offscreen buffer onto the active display buffer. This process continues until the end of simulation.

Animating the virtual-time clock simulation required a totally different approach. Here, simulation events could not trigger off corresponding animation events since the time frame is virtual. Instead, we adopted the following approach: the simulation writes every simulation event into a trace file stored in the database. The virtual-time clock animation thread then reads the trace file (after simulation ends) and performs animation. The virtual-time has to be converted to real-time for animation. There are methods to draw each simulation event. The advantage of using virtual-time clock simulation in animating is that one can fast forward, rewind, pause while watching the animation.

An added advantage of using the Web as the interface (Java applets) is that one has unlimited access to image sources and other resources needed for animation. For example, rather than having geometric shapes move around in a bank simulation, one could use images of peoples heads.

3.3 Database Access Facilities

The conceptual basis of integrating our simulation library with database access is QDS. Our simulation environment supports QDS in a natural and flexible manner. QDS is characterized by two basic ideas:

1) Storing simulation results and simulation models in a database: QDS emphasizes that simulation results, which are usually generated at a great computational cost must not be discarded, but instead be stored in a database. Simulation models are also required to be stored and retrieved systemically. Currently, we used the MiniSQL (mSQL) (Kimpton, T.R., 1995) database engine for our implementation. The reasons for using mSQL are as follows:

- mSQL, being a lightweight database engine that implements a subset of ANSI SQL, requires less memory and overhead than most relational DBMS engines. Nevertheless, it is powerful enough for our requirements.
- mSQL is the only DBMS which has a Java API, MsqlJava (Collins, D. 1996), at the moment this document is being written. We wanted database access to seamlessly integrate with the simulation library. This required that the DBMS provide an API rather than resorting to embedded SQL for database access.
- We expect that conversion from MsqlJava API to JDBC calls will be easy when JDBC is made available by JavaSoft and JDBC driver managers are made available by individual DBMS vendors.
- mSQL is free of cost and is available for download from http://Hughes.com.au/product/msql/.

We have designed our database to maintain a Model Index and store simulation results. The Model Index contains a list of all available models along with a brief description and information about the parameters the model requires (refer to Figure 1). It also contains a Uniform Resource Locator (URL) to the model applet. Each model is associated with a table in which it stores its results. These are the tables that are queried when a user issues either a normal query or a QDS query. The models, themselves, are stored as Java applets. These applets are invoked whenever a model needs to be executed.

2) Presenting a simple user interface: QDS envisages that even naive users must be able to use a simulation system based on QDS. We chose SQL as our user interface query language because SQL is both popular and simple. Currently, our user interface is functionally simple where the user can issue a query on one or more models (each model stored as a single table in the database) using one or more conditions. The available models and the stored data attributes can be obtained by querying the database. The models satisfying the query will be displayed to the user in a scrollable list. The user can select the model and the attributes he/she is interested in from the model list and can either run the model or issue a query. The user is also allowed to specify a conditional search which will serve as the where clause in the SQL query. The query is then sent to the database engine which will return the results if they exist in the database. If the results are not available in the database, the relevant simulation model is executed and the results are provided to the user.

The basic ideas of QDS also help overcome a problem posed by Java - that of providing persistence to simulation results. Java's in-built security restrictions as well as the security restrictions imposed by most Web browsers do not allow Java applets to write applet generated results into a file. We solved this problem by writing results into a database thereby shifting the responsibility of security from the browser to the database which often has elaborate security provisions. We have been successful in writing results from an applet into the mSQL database while working with the HotJava browser. The popular Web browser, Netscape, imposes a further restriction on database access by requiring that both the http server and the database server be running on the same machine.

3.4 The JSIM Class Library

The goals of creating the JSIM simulation library are:

- *to provide a set of utilities for building a working simulation*
  JSIM provides the basic utilities for creating a working simulation. Our implementation of a bank simulation illustrates the power of the simulation library.

- *to provide statistical or other data to be stored in the database*
  As a key component of query-driven simulation, the simulation component generates statistical data during model execution, which will then be stored in a database for future queries. As stated before, re-executing a simulation model with the same input parameters as before could be expensive if we do not store the previously generated data.

- *to provide facilities for animation*
  Java is a language for programming on the Web and is designed to deliver interactive content using applets embedded in HTML pages over a network of heterogeneous systems. Our simulation system can be used to animate the simulation process using applets.

- *to provide an instructional tool for simulation*
  The saying "it is better to see once than hear a hundred times" has been verified by psychological experiments (Humphreys and Riddoch, 1987). Visual facilities such as overhead projectors and models have been widely used for educational purposes and are proven to be effective, especially for explaining abstract concepts. For the people who are used to concrete thinking, this is also supported by research in interface design (Hutchins, Hollan and Norman, 1985). The rapid development of the Internet, especially the World Wide Web, places a huge impact on the way we live and learn. With animation, students are able to visualize and interact with simulation processes, which will help them learn better and reduce their learning curve.

- *to avoid learning a new simulation language*
  Java is an object-oriented programm-ing language and its syntax is similar to C++. Java is arguably simpler than C++ and is, therefore, easier to learn for novice programmers, easy to adopt for C++ programmers, yet powerful enough for complex tasks. With all its power and wide applications, Java becomes an alternative programming language to C/C++ and gains popularity with the rapid growth of Web technology. After having grasped Java, there is no need to learn other special simulation languages and simulationists can use JSIM to develop simulation models quickly.

The following are the classes in the JSIM library: *Display_Thread, Graph, Histogram, Plot, Model_Applet, Queue, FCFS_Queue, LCFS_Queue, Priority_Queue , Resource, Server, Simobject, Statistic, Transport* and *Variate*. Figure 2 shows the class inheritance hierarchy in JSIM. There are three types of queues derived from *Queue* in JSIM: FCFS_Queue, LCFS_Queue and Priority_Queue. The *Statistic* class implements methods for collecting, analyzing, testing, and displaying simulation results. The *Tally* method gathers sample statistics, while *Accumulate* gathers time-persistent statistics. In addition, the

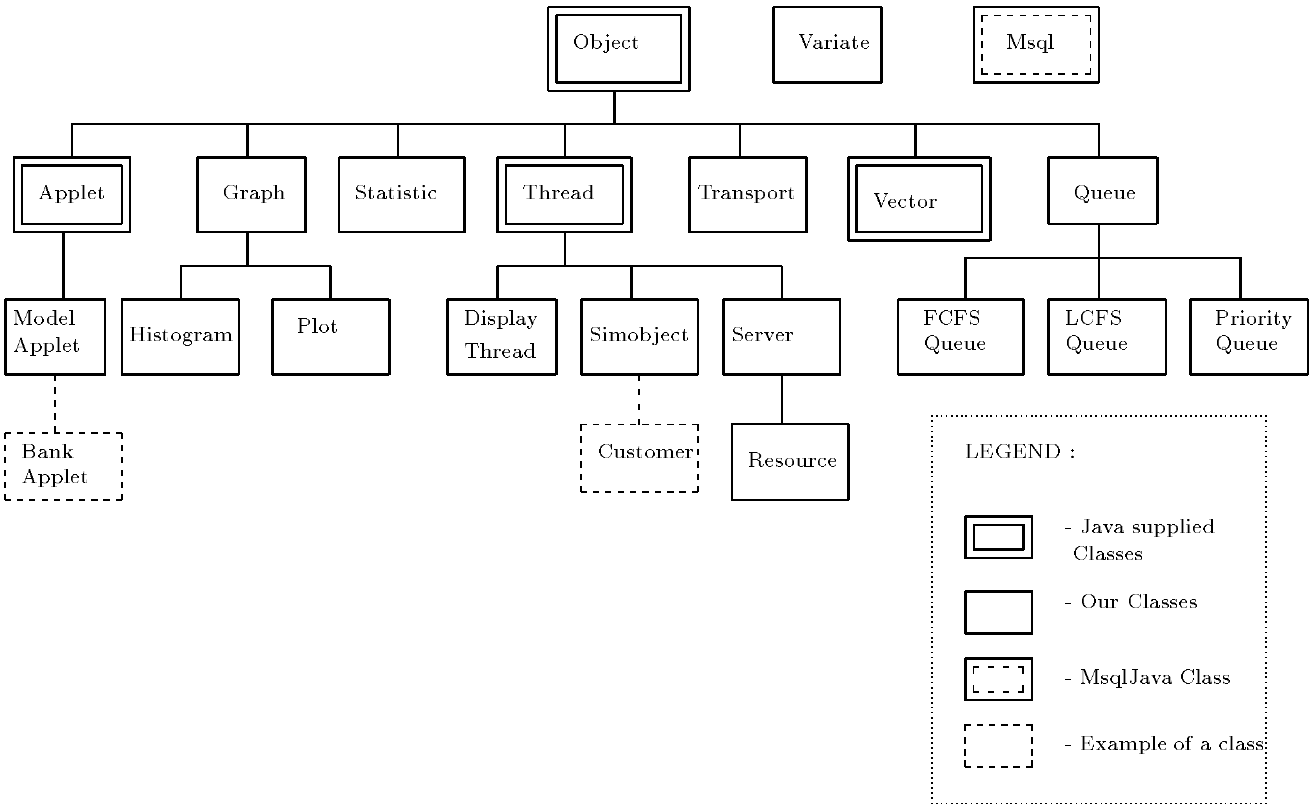

Figure 2: Class hierarchy in JSIM

Statistic class also provides methods for calculating mean, variance, standard deviation of sample statistics, e.g., mean of customer inter-arrival time. The class *Variate* provides procedures to generate random numbers and random variates. It has nine procedures for continuous random variates and four for discrete random variates.

The *Server* class, derived from *Thread*, is the service provider and has basic methods such as associating queues with the server and setting priorities to associated queues. The *Resource* class, derived from *Server,* allows user defined resources for simulation models. A resource consists of a number of service units (e.g., bank tellers), and a Queue feeding the service units. A process requests resources by calling *request*, and releases resources by calling *relinquish.* If a request for resources is not granted, the process making the request waits in the resource queue. The *Histogram* and *Plot* classes are responsible for generating histograms and plotting simulation results, respectively, and they are derived from *Graph*. The *Transport* class provides movement facilities to the simulation objects. Simulation objects can move only using transports or queues. The transport object is used to connect two key locations, for example, the head of a queue and a service station at a resource. The *Simobject* class provides an abstraction for simulation entities, for example, the customer class is derived from the Simobject class in the bank simulation example of Figure 3. The Simobject class houses basic methods for joining a queue, claiming a resource, etc. The *Display* class implements the display thread that is responsible for animating the simulation.

4 AN EXAMPLE

We use a classical bank simulation to show the power and capabilities of our simulation environment. The Bank_Applet is an aggregation of five types of objects: one or more Servers (Tellers), a FCFS_Queue, a Transport, a Customer_Generator and many Customers. The relationships between these components are shown in Figure 3 which is an OMT diagram (Rumbaugh et al., 1991).

The Bank_Applet models an M/M/s queue where s represents the parameter Num_Tellers. Both the Customer and Customer_Generator are derived from the Simobject class. The Customer_Generator class is used to generate several Customer threads based on the specified Inter_Arrival_Time. The Customer thread and the Customer_Generator thread make use of the Variate and Statistic classes to maintain their

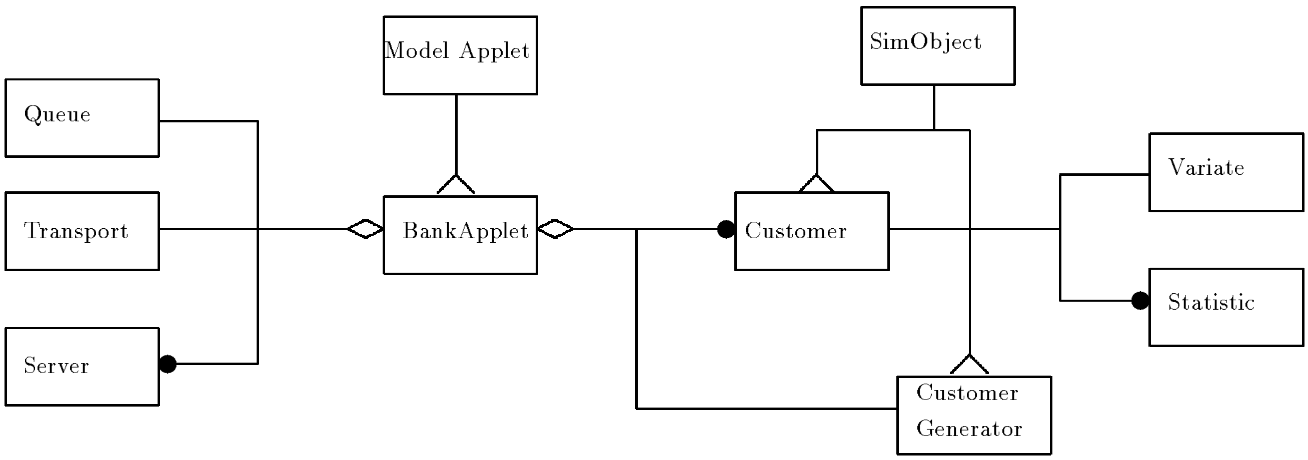

Figure 3 : Bank Simulation Example

statistics. The Transport class is used to connect the Queue to the service station at the Server.

When a Customer entity is first created, it tries to engage the Server. If the Server is busy with some other Customer, the Customer entity enters the Queue and waits its turn. If the Customer is able to engage the Server, it will be served for a period of time referred to as service time which is derived from an exponential distribution. After the service time is over, the Customer relinquishes the Server and leaves the bank (Customer thread terminates).

A typical QDS query for the above bank simulation model would be as follows:

```
SELECT Waiting_Time, Avg_Queue_Length
FROM Bank_Applet
WHERE Num_Tellers=1 AND Inter_Arrival_Time=3
```

In response to this query, the table corresponding to the bank model will be queried and the results returned if available. If the required results are not available in the table, then the QDS system retrieves the model information from the Model Index table. It then invokes the relevant model applet using the parameters given by the user in the query, namely, Num_Tellers = 1 and Inter_Arrival_Time = 3 (other parameters will take on their default values). The user is then given the results of this new model execution.

5 CONCLUSIONS AND FUTURE WORK

We believe that Web-based simulation will become an important technology for the future. The importance of the World Wide Web steadily increases; and now with Java, highly interactive and dynamic simulations/animations may be executed on the Web. Recognizing this, we have an ongoing project to develop sophisticated capabilities for Web-based simulation. Our QDS environment and JSIM library provide a widely available and easy-to-use facility for developing Web-based simulations and animations. The environment combines simulation, animation and database access exploiting the strengths of Java (e.g. it is a high-level, clean and safe object-oriented language and has multi-threading and sophisticated display/GUI facilities) as well as minimizing its weaknesses (e.g. lack of data persistence in applets).

As an ongoing project, we still have much work to do. The following are the major areas of interest for future work. We are currently incorporating advanced features such as batch/unbatch, split/join and routing features into our library. We will also be converting from MsqlJava API to JDBC API when a stable version of JDBC becomes available. This will allow us to use a variety of Relational or Object-Relational DBMSs. Another area of focus is the Graphical Model Designer. The Graphical Model Designer is intended to be a GUI- based model builder that a simulationist can use to quickly build models by drag and drop operations.

ACKNOWLEDGEMENTS

We would like to acknowledge Darry Collins for writing and making available the Java API to mSQL. We would also like to thank Vijayalakhsmi Natarajan, Department of Computer Science, University of

Georgia, who helped us in developing database access for our class library and Hongwei Zhao, Department of Computer Science, University of Georgia for contributing to the design of the graphical designer.

REFERENCES

Collins, D. (1996) *MsqlJava: A Java class library for mSQL.* URL: http://mama.minmet.uq.oz.au/msqljava/tutorial.html

Hutchins, E.L., Hollan, J.D. and Norman, D.A. (1985). Direct manipulation interfaces. *Human-Computer Interaction.* 1:311-38.

Humphreys, G.W. and Riddoch, M.J. (Editors) (1987). *Visual Object Processing: A Cognitive Neuropsychological Approach.* Hove, UK: Lawrence Erlbaum Associates.

Kimpton, T.R. (1995) *Mini SQL: A lightweight database server.* URL: http://Hughes.com.au/product/msql/manual.htm

Miller, J.A., Kochut, K.J., Potter, W.D., Ucar, E., and Keskin, A. (1991b). Query-driven simulation using active KDL: A functional object-oriented database system. *International Journal in Computer Simulation,* 1(1): pp. 1-30.

Miller, J.A., Potter, W.D., Kochut, K.J., Keskin, A., and Ucar, E. (1991a). The active KDL object-oriented database system and its application to simulation support. *Journal of Object-Oriented Programming,* 4(4): pp. 30-45.

Miller, J.A., Potter, W.D., Kochut, K.J., and Ramesh, D. (1996a). Object-oriented simulation languages and environment: a four-level architecture. *Object-oriented Simulation.* Zobrist-Leonard, Editors. IEEE Press (1996). (to appear)

Miller, J.A. and Weyrich, O.R. (1989). Query driven simulation using SIMODULA. *Proceedings of the 22nd Annual Simulation Symposium, Tampa, Florida.* pp. 167-181.

Miller, J.A., Weyrich, O.R., Potter, W.D., and Kessler, V.C. (1996b). The SIMODULA/OBJECTR query driven simulation support environment. *Progress in Simulation,* Vol. 3. Leonard-Zobrist, Editors. (to appear)

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991) *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey.

AUTHOR BIOGRAPHIES

RAJESH S. NAIR is a graduate student in the MS program in the Department of Computer Science at the University of Georgia. He was awarded a University-wide Assistantship for the year 1995-'96. He received his BE degree in Computer Technology from the University of Bombay, India in 1994. His research interests include application of database systems and simulation environments. This paper is part of his ongoing research work toward the MS program.

JOHN A. MILLER is an Associate Professor and the Graduate Coordinator in the Department of Computer Science at the University of Georgia. His research interests include Simulation, Database Systems, and Parallel & Distributed Systems. Dr. Miller received the BS degree in Applied Mathematics from Northwestern University in 1980, and the MS and PhD 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 has been active in conferences in both simulation and database. He has been the Publication Co-Chair for RIDE (Research Issues in Data Engineering), President/General Chair of the Annual Simulation Symposium, and Coordinator of the Modeling Methodologies Track of the Winter Simulation Conference. He has also been a Guest Editor for the International Journal in Computer Simulation.

ZHIWEI ZHANG is a graduate student in the MS program at the Department of Computer Science, University of Georgia. He received his Masters in Library Science from the University of Maryland and a certificate in Library Science from Nanjing University, China. His research interests include Simulation and Database & Information systems.