Netsim: A Java™ -Based WWW Simulation Package

by
Tamie Lynne Veith

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Industrial and Systems Engineering

Approved:
Dr. Pat Koelling, Co-chair
Dr. John Kobza, Co-chair
Dr. J.W. Schmidt

April 29, 1997
Blacksburg, Virginia

Keywords: Internet, Java, modeling, Netsim, programming, simulation, WWW

Netsim: A Java™ -Based WWW Simulation Package

by
Tamie Lynne Veith

(ABSTRACT)

Use of the World Wide Web (WWW) for transfer of information and ideas is increasingly popular. Java, a programming language for the WWW, provides a simple method of distributing platform-independent, executable programs over the WWW. Such programs allow the expansion of WWW-based computational and analytical tools that support and enhance the existing WWW environment. However, a WWW-based, generalized simulation package is not yet available. Current literature motivates development of a general, WWW-based simulation package with maximum user interactivity and cross-platform capabilities. Advantages of such a package are discussed and explored in three potential applications. Main advantages are wide availability, controlled access, efficient maintenance, and increased integration. Disadvantages, such as variable download times, are also discussed. Netsim, a general, WWW-based simulation package written entirely in Java, is developed and demonstrated. Netsim provides complete model creation and modification capabilities along with graphical animation and data output. Netsim uses the event graph paradigm and object-oriented programming. Java, event graphs and object-oriented programming are discussed briefly. The Java random number generator is verified for uniformity and independence. Netsim is compared to SIGMA, a non-Internet simulation package, using a standard M/M/1 queueing model. Comparison issues and results are discussed. Additionally, tested through hand-tracing for coding validity, Netsim performs as theory prescribes. Netsim documentation and user's manual are included. Netsim allows expandability for complex modeling and integration with other Java-based programs, such as graphing and analysis packages. Current Netsim limitations and potential customization and expansion issues are explored. Future work in WWW-based simulation is suggested.

Acknowledgments

I would like to acknowledge my committee's time, advice, and dedication to this project. In particular, my thanks to Dr. Kobza for his help, throughout the summer, in formulating and motivating the project proposal. Additionally, my sincerest thanks to Dr. Koelling for his past year and a half of availability and support, including funding and work space, even while on a semester sabbatical.

Also, I would like to thank Ron and Theresa for ensuring my familiarity with the ETD process and for making sure I stopped to eat occasionally.

Finally, my deepest gratitude to Archer for her constant support, understanding and constructive suggestions throughout the past two years.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

A growing number of people use the Internet daily for both business and pleasure. The Internet is a computer network through which many types of data can be transferred. The vast system of server computers connected to the Internet, the development of standardized protocols, and the Uniform Resource Locator computer addressing system have helped create a worldwide information system within the Internet, called the World Wide Web.

People increasingly connect to the World Wide Web (WWW), through WWW browsers such as Netscape Navigator (Netscape 1997), to access resources, share information, schedule meetings, and leave messages. Through Java™ (Sun, 1997; Appendix A), a WWW-based programming language, people can also customize and interact with WWW-based computational programs as they would with traditional, non-Internet-based programs. However, WWW-based applications do not yet include a generalized, interactive simulation program for modeling, demonstrating, and analyzing a particular project. As a result, many WWW users must still use a specialized, non-Internet-based simulation program to work with the technical aspects of a project.

A general, WWW-based simulation package can enable users to model, explore and analyze the same problems for which they would otherwise need traditional simulation packages. The resulting models are then easily explored and manipulated by any colleagues or clients connected to the WWW without the introduction of additional, possibly unfamiliar software. Because Java programs operate on WWW browsers, they are platform-independent, thus avoiding system compatibility problems often encountered with non-Internet programs. Additionally, WWW applications are generally maintained in a single location, allowing the provider to easily distribute modifications, maintain version control and control user access. Other benefits with WWW-based applications include access from distant sites and off-hour availability.

The purpose of this research is to create and demonstrate a general, Java-based, WWW simulation package called Netsim. A simulation problem, modeled in Netsim, can be instantly viewed as an animated model on a WWW browser, allowing users to benefit fully from the advantages of the WWW environment. Such Java-based models allow the user the same degree of interactive, multimedia capability as do non-Internet-based programs (Jones, 1996). Additionally, the object-oriented paradigm used in Java-based WWW simulation provides distinct modeling advantages over more traditional procedural-based simulation packages such as SLAM II® (Pritsker, 1986; Appendix A) or SIGMA (Schruben, 1995; Appendix A).

Chapter 2 outlines and discusses the advantages and disadvantages of a WWW-based simulation package, such as Netsim. In addition, three applications in which Netsim could be advantageous are considered. Chapter 3 presents background information on Java, object oriented

programming, and event graph simulation. A literature review of Internet, WWW and simulation-based work leading up to this project, as well as current work in this rapidly evolving area, is presented in Chapter 4. Chapter 5 motivates the development of the Netsim simulation package, discussing the ways Netsim answers limitations of and contributes significantly to the existing set of WWW-based simulation tools. Netsim's structure and capabilities are explained in Chapter 6. Chapter 7 describes the procedures used for verifying the accuracy of Netsim and also reports and analyzes the results of this verification. Conclusions and ideas for future research are given in Chapter 8.

Chapter 2: WWW-Based Simulation

2.1 Advantages and Disadvantages

As mentioned in the introductory chapter, a WWW-based simulation program provides several beneficial features that are lacking in currently established, non-Internet-based packages such as SLAM II or SIGMA (Nair, 1996). These features include wide availability among different systems, controlled access, efficient maintenance, and increased integration into current working environments. A few disadvantages also exist when simulating over the WWW. These include difficulty in tailoring a program to a specific platform, loading time variability, and the possibility of inconsistent model access. The following sections discuss advantages and disadvantages of WWW-based simulation.

2.1.1 Advantages

Advantages of a WWW-based simulation program can be grouped into the following main characteristics with associated features:

Wide Availability

- allows access on many platforms without recompiling.
- allows access at distant sites without transporting hardware or software.
- allows access outside normal business hours.

Controlled Access

- protects against inadvertent and unauthorized change and duplication of original.
- allows “copy exactly” model distribution.
- enables individualized access through passwords on unrestricted machines.
- enables limited time-span access.

Efficient Maintenance

- enables frequent modifications to be made and instantly distributed.
- reduces error potential when updating and distributing models.
- eliminates virtually all on-site maintenance.
- allows modifications and implementations to be made through the server.

Increased Integration

- interfaces instantly with existing WWW browsers.
- requires only a WWW browser capable of viewing Java programs.
- encourages communication and interaction through the WWW.

Wide Availability. Platform specificity or recompilation is often necessary with non-Internet-based simulation software. Utilizing traditional simulation packages requires access to a

computer containing the proper simulation software, as well as an appropriately compiled copy of the model source code. The portability of a WWW-based model onto numerous platforms enables the user to quickly access and run the model from multiple, wide-spread locations. Users at distant sites can instantly access the most up-to-date models using the available computer platform. There is no need to transport hardware or software to these sites, download an appropriate version of the file, or recompile the code. Furthermore, since the Internet is usually available twenty-four hours a day, access to a model and its WWW site is not limited by time constraints. This allows users to proceed at their own pace and to work within their own time schedule.

Controlled Access. Permanently modifying a model in either WWW-based or traditional simulation packages requires access to the model source code. With non-Internet simulation packages, the user often accesses the model directly from the computer on which the model is running, with the complete source code of the model and the software package residing on that computer. In this situation, there is significant potential for confusion and later difficulties caused by inadvertent, permanent changes to and multiple copies of a model. Additionally, users are able to retain, change, duplicate, and share models or software beyond the limits of any agreements they might have made with the supplying company.

On the WWW, a single copy of any model, provided through a server, is necessary. Also, although Java-based WWW simulation packages, such as Netsim, are accessible through any Java-compatible browser, the package may not be copied without direct access to the uncompiled package source code. As a result, only a person with access to the original model or package code can permanently change the model or allow duplication of the simulation package, preventing inadvertent or unauthorized alterations and the resulting difficulties. This can be particularly useful for companies providing “copy exactly” instructions through a model and depending on exact replicas of a product based on information from that model.

When using WWW-based simulation, one person can control access to a model through secure WWW sites, password requirements and limited time-spans. This helps link the use of the model to the appropriate people as opposed to specific computers with multiple users, as often happens with current simulation packages. Additionally, such access restrictions are placed on the model or the WWW site, not on the machine itself. This ensures uniform access privileges regardless of platform.

Efficient Maintenance. With WWW-based simulation the existence of a single working model enables frequent, permanent modifications with a smaller degree of error. Additionally, there is seldom need for on-site maintenance. The model or simulation package can be modified and made instantly available over the WWW and users notified, if necessary, through e-mail. In the more traditional packages, implementing modifications can be tedious and time consuming. First, users

must be made aware of the need for and existence of an updated model. Then the update must either be made on each copy of the model, increasing the chance of model inconsistencies, or the new model be delivered to and reinstalled onto each computer and the old one deleted.

Additionally, the creator/ maintainer of a WWW-based simulation model generally has access, through the Internet, to the model's source code on the serving computer. This allows that person to access the server from a distant site, if desired, in order to permanently modify the model. For most traditional simulation packages, any changes to the source code must be made directly on, or copied to, the computer running the simulation model.

Increased Integration. Programming software and viewing browsers that support a WWW-based simulation package are readily available through the Internet. Appropriate browsers, such as Netscape, are Java-compatible and already installed on many computer systems. Where not currently installed, these browsers are easy to locate and available for immediate installation. Many such browsers are currently free to educators, researchers, and students or are included in standard office software packages.

Since many users are familiar with navigating WWW browsers, the total software learning curve for using a model is minimal. Once the browsers are installed, no additional software or downtime is required before accessing a simulation model. Unlike many standard simulation packages, the needs for proper installation of the software package and for working knowledge of the operating commands do not pose major hurdles to effective, timely use of a model. Additionally, the WWW-based software can be made user friendly, enabling the creation of non-technical, menu-driven models. This encourages users to become more involved with the available technology and minimizes the need for assistance by a skilled programmer.

Active use of the WWW allows the user to immediately and efficiently resolve issues raised by a simulation model. Each WWW simulation model is linked to a hypertext markup language (HTML) page. This page can contain e-mail links, help or problem request forms, connections to relevant search engines, and links to previous models. Such connections are easy for a WWW site programmer to provide, maintain and use. Additionally, the page can provide supplemental information in the form of textual instructions, images, sound clips, and other appropriate multimedia tools. User communication might be further enhanced by Internet tools such as list-serves, electronic bulletin boards, or chat rooms. While examining a model, a user can easily send messages, answer questions, explore a subject area and research a particular topic in depth. There is no need to exit and reopen software as might be necessary when using more traditional simulation packages.

2.1.2 Disadvantages

Simulation over the WWW does have a few potential drawbacks. These disadvantages, as well as

the previously discussed advantages, should be considered when deciding which simulation package best meets a user's needs. Users not relying on the possibilities available through WWW-based simulation or not interested in connecting to the Internet might find a traditional package more suitable, as might users frequently modeling large or complex systems.

The following explain benefits of traditional simulation packages over WWW-based ones.

- Many current simulation packages are each specialized to idiosyncrasies of a single platform, making maximum use of its capabilities. This may increase efficiency of simulation runs when using models that require few or no updates over long periods of time.
- Loading times for traditional programs are dependent on the computer and not on the current volume of Internet usage. During times of heavy Internet traffic, models with complex or extensive amounts of code may initially take a large amount of time to download.
- Because each computer usually contains a copy of the traditional simulation software, the user can generally rely on the availability of the software. When using the Internet, however, temporary interruptions in Internet service may cause the WWW-based simulation model to be momentarily unavailable.

2.2 Applications

The advantages of WWW-based simulation suggest far-reaching applications involving timely, long distance interactions or temporary, wide-spread viewing of models. Researchers, instructors and organizations can share timely simulations of their current work with partners, students, clients, and potential customers over the WWW. Anyone connected to and familiar with the WWW already has access to and knowledge of the appropriate software. Additionally, viewing access to the most recent model is easily arranged and maintained.

Researchers can use an on-line simulation program to share up-to-date work with collaborators and with grant providers. Instructors of distance learning classes can use WWW-based simulation models to provide demonstrations and hands-on examples and to encourage student feedback. Manufacturing companies can increase communication between plants by providing immediate, visual demonstrations of the processing layout for a new product. Other companies might use such simulations to advertise new products to clients. Consultation and management advisory groups can use WWW-based simulations to provide role-playing games and problem-solving services.

The following examples outline the unique benefits of WWW simulation modeling as it pertains to three types of situations: competitive business performance, marketing and product sales, and

skill training and role-playing. Additionally, these examples go beyond the basic queueing models currently provided by Netsim to explore future possibilities of WWW simulation.

2.2.1 Business Performance

In today's fast-paced, corporate world, businesses try hard to keep ahead of competitors and of the market. To do so often involves major changes, such as reorganization of management structure and cutbacks in personnel. Managers of an organization must be able to determine how such changes will affect the company and its outside interactions. Through WWW-based simulation, companies can make timely, knowledgeable decisions with regard to major organizational restructuring for their company.

With WWW-based simulation managers can:

- access the simulation while at distant companies or collaborating sites.
- always access the model version reflecting the most current situation.
- instantly access the model through any Java-compatible, WWW-linked computer.

Outside collaborators and physically distant areas of the company can become actively involved in the situation of the company. WWW-based simulation eliminates the need to carry along computers with appropriate software and latest-model versions. Representatives of the company visiting these collaborators or other companies can easily demonstrate the active improvement of their company by accessing the simulation model over the WWW.

A WWW-based simulation model can reflect the current state of the particular company and of outside influences (competitors, customers, world and local markets). The model parameters can be updated frequently as the company or outside influences change; for example, as other companies restructure or markets fall. A mutual fund company might use daily updates of a stockmarket econometric model to forecast interest rates and update portfolios. Fund managers can then access these updates twenty-four hours a day to determine buy/sell actions and advise investors.

A company using a traditional simulation program must submit a change in the model to the programmer and then wait for the updated model to be distributed and installed. By placing the updated model on a WWW server, the programmer can significantly reduce this waiting time. The client/server relationship of the WWW makes the updated models instantly accessible, allowing managers to make decisions with regard to the latest information about crucial factors.

2.2.2 Product Sales

Timely demonstrations of the benefits of new products can be vital to the growth of competing

companies, both supplier and consumer. Customer feedback regarding a supplying company's products can help the marketer focus its efforts. Likewise, easily accessible, well-explained product previews help a consumer company function and expand smoothly. Demonstrating, via an WWW-based simulation model, the ways a new product can enhance the customer's system is effective for both the marketing company and the customer and encourages suppliers to market new products in a cost-effective, consumer-aware manner. For example, a production-line machine supplier might use WWW-based simulation modeling to demonstrate a more efficient machine. By providing two simulation models, one based on the current machine and one on the new product, and by allowing customers to input model parameters for their particular production runs, suppliers can easily demonstrate the benefits of the new products.

Using WWW-based simulation to display new products, the marketing company can:

- make previews instantly available to anyone with a Java-compatible WWW browser.
- make one copy widely available through multiple platforms to potential customers.
- utilize the WWW to provide on-line assistance and receive customer feedback.

By making a single copy of the simulation model available over the WWW instead of mailing multiple copies, the company can reach more people, especially individuals and small businesses who might otherwise be overlooked. The demo is instantly available to anyone using a Java-compatible WWW browser. This method eliminates the need for costly copies of demo software and instructions for viewing the product. Additionally, the WWW-based simulation models are platform-independent, assuring the customer will be able to view the model on any available Internet-connected machine with a Java-compatible browser.

E-mail and other relevant WWW links provided along with the demo can help customers learn about the company, locate additional products, and direct questions to the most helpful source. Likewise, survey forms and search engines built into the preview help the marketer find out more about the needs and concerns of their customers through customer feedback, orders, and complaints and can allow the marketer to respond quickly on an individualized basis. Combining these types of WWW-based tools with simulation models describing the product provides an interactive connection between the marketer and both potential and existing customers.

2.2.3 Skill Training, Role-playing

Many companies are reducing their levels of management and increasing their use of teams throughout the leadership hierarchy. As the company hierarchy changes, managers may need additional education and training to effectively carry out their new responsibilities. They will need insight into the roles of other managers to work most effectively as a team. To help with these tasks, situational and role-playing games may be used. These games simulate the behavior of an organization and the positions of the people within the organization. Additionally, they

allow users to perfect skills and role play as the need arises to resolve conflicts or uncertainties in a timely fashion.

Frequent use of simulation models in this manner can lead to closer knit teams with the ability to improve their own operations. Such use can also heighten awareness throughout all levels of the organization as to interactions between and contributions of its various members. Skill training and role-playing via WWW-based simulation provides important advantages over non-Internet based simulation, particularly in terms of maintenance and access control.

A WWW-based training or role-playing model can:

- allow learners to access the model as their schedule permits.
- allow providers to control the level of user access and modification to the model.
- allow providers to address and resolve problems through the providing server.

Because the WWW is usually available 24-hours a day, users can access the training or role-playing model as their schedule permits. Their access is not limited to normal business hours, scheduled around the availability of a computer with the necessary software. This allows users to work at their own pace without the pressure of finishing within a given time slot.

Through the server the provider can control access to skill training and role-playing models by including passwords for the site or particular models and by making the model available for a set time period. Additionally, the use of Java in developing the models helps protect them from user changes and duplication. This ensures the provider that the fair use guidelines agreed upon by the user will be upheld. These same features allow many people in a company to work simultaneously with a model. For example, users can role play in teams or individually to promote understanding or resolve conflicts. Additionally, users need not worry about making inadvertent changes to the model or maintaining copies of the model on their local machines.

Utilizing the server/client relationship of WWW-linked computers allows the provider to permanently revise a model through connection to the server computer. This eliminates visits to the users' actual locations, as required by non-Internet simulation models. In addition, it helps enable problem resolution in a timely manner and frees the user from downloading and maintaining files.

Chapter 3: Structural Background

Netsim uses the event graph simulation method and the object-oriented structure of the Java programming language. This combination allows the capabilities of Netsim to be easily modified, maintained, or expanded and allows for code re-use. Additionally, it helps minimize the amount of computer memory used by a Netsim simulation model. This chapter provides background information about Java, object-oriented programming, and event graphs.

3.1 Java™

Development of Java, as an object-oriented programming language, began in 1991 by a team of programmers at Sun Microsystems, Inc. Their goal was to use Java in creating fast, platform-independent software that could be used in simple, consumer electronic products. As a result of Java being designed for simple, efficient, platform-independent programming, it appeared to be an ideal language for creating WWW-based programs. In 1994, as the WWW became increasingly popular, the Java team began to modify the Java language to work through the WWW and created the first Java browser, WebRunner (December, 1995). Interest caught on to the capabilities of Java for the WWW, and Sun produced a more stable Java browser, HotJava, as well as a Java Development Kit. Many WWW browsers are now Java-compatible, and there are a growing number of software packages for developing Java programs.

Using Java one can create small programs called applets that are embedded into an HTML document and viewable on any Java-compatible browser. Java applets are compiled into a set of bytecodes, or machine-independent processing instructions. This set of bytecodes is then translated by an interpreter within a Java-compatible browser and the applet is executed (Lemay and Perkins, 1996). Java's high portability enables an applet to be accessed through numerous platforms on almost any computer connected to the Internet.

Java's programming language closely resembles the C and C++ languages with a few alterations, such as the removal of pointers, specifically intended to make Java more robust. This similarity makes switching from C++ to Java similar, in difficulty, to switching to an updated version of C++ (Freeman and Ince, 1996; Aitken, 1996). Java further simplifies use for the end user, as well as the programmer, by supporting graphical user interfaces and a number of menu formats.

Java utilizes complete object-oriented programming, a programming technique particularly well-suited for simulation modeling (Banks, 1996). Netsim takes advantage of this object-oriented structure.

3.2 Object-Oriented Programming

Object-oriented programming languages reduce the need to pass multiple parameters sequentially through a program as is often necessary in procedural languages such as Pascal or FORTRAN. Instead the program consists of a number of *objects*. Each object is a module defining some aspect of the program's process. As the program runs, the objects interact as necessary by passing parameters back and forth. By using separate modules to define unique attributes, object-oriented programming allows easy modification and code reuse. This results in efficient, cost-effective model creation and maintenance (Freeman and Ince, 1996; Joines and Roberts, 1996).

Object oriented languages consist of several *packages*, or logical groupings of *classes*. A class is a group of program coding that defines a set of attributes and behaviors. For example, one class will define the features needed to display and maintain the user interface, while another will enable running a particular simulation model. Each class created in Netsim incorporates or expands on a number of standard classes contained in the Java class library through a process called inheritance. For example, the Netsim class defining the viewing interface includes the java.awt.Graphics class and extends the java.awt.Canvas class. These classes provide behaviors for drawing, updating, and displaying the graphic version of the simulation model onto the interface. Both of these classes are contained in the Abstract Window Toolkit (AWT) package of the Java class library, available to all programmers.

Activating a particular class creates an *instance*, or *object*, of that class. The class acts as a template, with its attributes and behaviors defining the instance. For example, while Netsim's viewing interface class defines the interface, an instance of that class represents those definitions at any particular step of the simulation. As the user runs a simulation model, instances of classes are created and destroyed on an as-needed basis.

Figure 1 demonstrates the way Netsim uses the inheritance structure of object-oriented languages. Three object-oriented packages are represented in this figure: java.lang, java.awt, and netsim. The solid arrows depict subclass relationships, i.e., netsim.SchedThread is a subclass of java.awt.Thread which is a subclass of java.lang.Object. By this subclassing, an instance, or object, of netsim.SchedThread contains all the behaviors defined by each of these three classes.

Each object-oriented class contains a number of *methods* that analyze and manipulate various aspects of a current instance of that class. When a user initiates a simulation run, the program coding creates instances of classes and requests the instances call appropriate methods within their associated classes. These methods either provide information about that instance, such as size or color, or change an aspect of the instance, often by performing a set of operations such as reordering variables in the instance array. In Figure 1 the dotted arrows depict the types of messages passed between Netsim classes via methods.

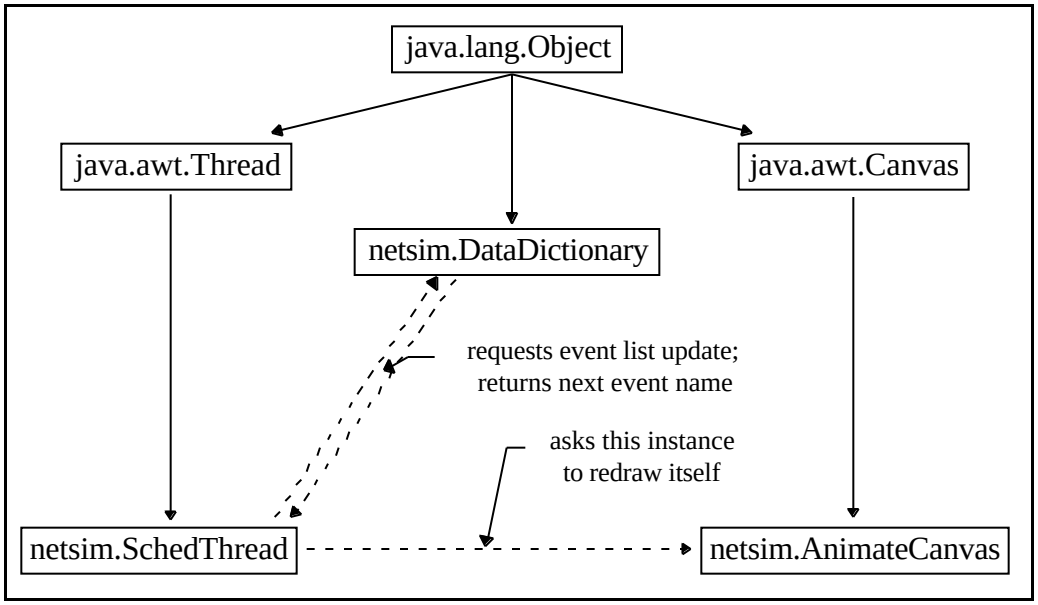

Figure 1: Example of object-oriented hierarchy

3.3 Event Graphs

Event graphs are a means of capturing the behavior of a simulation model in a modular, graphical manner (Buss, 1996; Schruben, 1995). This section briefly discusses event graphs and their connection with object-oriented programming. For a more detailed description of event graphs see the SIGMA documentation (Schruben, 1995).

Event graphs consist of two basic elements, event nodes and connecting edges. The nodes represent moments within the system during which one or more state variables, or event attributes, change. Simulated time passes on the connecting edges between nodes, provided any conditions associated with the respective edges are satisfied.

A future events list is used to keep track of scheduled events; as a scheduled event is activated, it is removed from the list and all associated state variables are updated. Then conditions on all edges connecting from that event are checked. If the conditions are satisfied and there is no time delay associated with that edge, the connecting event is scheduled immediately. Otherwise, given the conditions are satisfied, the time for the connecting event is determined and the event is added to the future events list. Because a simulation model, unlike a real system, can process only one event at a time, scheduling priorities are associated with the event scheduled by each edge. Events scheduled to occur simultaneously are placed on the future events list in order of their assigned priorities. By assigning priorities carefully, users can design models that more closely describe a system where events often occur simultaneously.

By adding simplicity and uniformity to the modeling process, event graphs allow users to focus on the creation and understanding of the actual models. Additionally, because they separate all the changes of the system into discrete nodes, or modules, they interface well with object-oriented programming languages. An event graph models a system visually, providing a global view of the system. The object-oriented program of the model transfers the visual nodes into objects with methods, or behaviors, among which the messages provided by the edge conditions are passed.

Chapter 4: Literature Review

In the 1980's Jack Thorpe, in conjunction with the United States Department of Defense, created SIMulator NETworking (SIMNET), a networked system of computers running a single simulation program. SIMNET was an attempt to make the use of simulators and simulation techniques more feasible for military defense operations (Fullford, 1996). In this system each user computer acts as a simulator, allowing multiple users to experience a single simulation program simultaneously in a safe, cost-effective environment.

The success of SIMNET resulted in a standardized simulation networking protocol, Distributed Interactive Simulation (DIS). DIS allows computerized simulators to communicate and act in synchronicity through the Internet. Although DIS began in the military environment, it is now being used increasingly often in non-military applications such as air traffic control, intelligent vehicle highway systems, and interactive, multi-user computer games. Work is underway to incorporate some aspects of Internet markup and programming languages such as Virtual Reality Markup Language (VRML) and Java to the DIS protocol. This will increase the abilities of DIS systems to define behavioral information and visual representations as users enter and exit the simulation environment.

There are several benefits of using not just the Internet, but the WWW and its existing structure of interconnected hypermedia, to enhance simulation technology (Neilson et al., 1996). Irene Neilson discusses the use of the Interact Simulation Environment (ISE), created as an aid to teaching engineering students. The ISE embeds simulation models into a graphical user interface while specific commands are placed into related HTML documents. The HTML commands, when activated, access a database that prompts an action from the simulation model. Also, at some simulation states, commands within the graphical user interface link the user to relevant HTML documents for more information. As a result, ISE supplements HTML articles with simulation demonstrations and allows context-sensitive help while viewing the simulation.

Because of differences in students' background computer knowledge, the ISE is set up to utilize simulation as a modeling tool without requiring the student to interact with the actual simulation interface. This way the students do not spend time away from their engineering courses learning the C++ classes that ISE uses to build the graphical interface for the simulation model. A student may save "snapshots" of the simulation model at a given state for later reference or to use in discussion with others.

While Neilson discusses many advantages of the ISE and the WWW teaching lessons that utilize simulation modeling, she does state a limiting factor readily overcome by the use of the Java language. Many WWW browsers use established protocols when interpreting data types and can

not interpret recently created or non-traditional protocols. As a result, the ISE, as currently developed, requires the original simulation model to be stored directly on the user's machine in order to access "snapshots" of that model. Java-compatible browsers, however, understand traditional, established protocols in addition to interpreting Java programs into executable programs. Using Java to develop the ISE would allow platform independent model interpretation and eliminate the need for multiple storage of the simulation, as well as the resulting increase in file maintenance.

Rodney Cole and Scott Tooker (Cole and Tooker, 1996) have developed WWW-based physics tutorials to assist physics students. Making these simulation models available over the WWW greatly expands the range of possible access locations. Like ISE, the physics tutorials allow students to see interesting cases of a given simulation model without requiring prior knowledge of the parameters defining these cases or of the background programming languages involved. Additionally, the use of familiar WWW browsers such as Netscape virtually eliminates the amount of time necessary for distributing and learning viewing software.

The tutorials use Apple's OpenDoc Frameworks to provide a basic simulation environment. OpenDoc Frameworks is a tool for creating software components, in this case physics simulation models. The components can then be included in an OpenDoc application that is embedded into a WWW page using HTML. OpenDoc supports modular development of models, allowing existing tutorials to be tailored to meet the needs of different educational grade levels**.**

Cole and Tooker state that future tutorials will consist of OpenDoc components, HTML, and multimedia effects created using Java. OpenDoc and Java are complementary tools (Apple, 1997); OpenDoc components can be written entirely in the Java language or Java applets can be embedded into the components. However, using OpenDoc in this method voids the platform-independence of Java. OpenDoc applications and OpenDoc components, such as those created by OpenDoc Frameworks, must be downloaded onto the user machine with a working copy of OpenDoc. Additionally, OpenDoc is currently available only for the Macintosh platform.

The importance of the WWW as a platform for interactive learning is supported by T. Singh (Singh et al., 1996). In particular, he discusses the WWW in connection with the added cross-platform benefits of Java. Singh demonstrates Java applets for teaching electromagnetics and theories of dipole-antenna. These tutorial applets are embedded in instructive HTML pages and allow some degree of user interaction with the presented models.

Gordon Bradley (Bradley, 1996) talks at length about using Java to share research ideas in a timely fashion over the WWW. He is particularly interested in the comprehensive value that Java applets can add to interactive, electronic research papers. Such papers are dynamic, report research more effectively, and allow non-linear evaluation, he claims. The inclusion of Java

applets can allow complete reproducibility of research results by allowing the author to embed the data, algorithms, and analysis work into the electronic paper. A reader may then instantly examine and manipulate these data without necessarily downloading data or software onto his/her machine.

Paul Fishwick (Fishwick, 1996) considers three important areas of educational WWW-based interaction: education and training, publications, and simulation programs. Using the WWW for education and training allows and encourages the reuse of knowledge while not limiting the amount of storage space available for expressing new ideas, as is the case with CD-ROMs or hard drives. Additionally, Fishwick discusses timesaving aspects of using WWW tools in reading and publishing research articles. The WWW is valuable in all stages of producing an article, from accessing documents via electronic databases or URL's cited in bibliographies, to transmitting electronic copies of the publication to reviewers and for final publication. Simulation models included in the electronic research publications enhance their educational value. The paper gives an example of such an electronic article where a simulation model using Perl script is included to demonstrate current, workstation resource, queueing sizes based on user input gathered through an HTML form. Besides the value of simulations in aiding understanding, Fishwick discusses the possibilities of WWW-based simulation or simulation interfaces for multi-user situations such as multi-user dungeons, where users typically cooperate to reach a solution, and DIS as discussed earlier.

Using simulation models based on the process interaction paradigm, Rajesh Nair demonstrates WWW-based simulation through a unique combination of Java applets and query driven databases (Nair et al., 1996). Actions by the user send queries to a database which then initializes the appropriate applet. The database stores data created by the running simulation and supplies these data back to the applet as necessary throughout the simulation run. Using a database in this manner helps circumvent some security issues currently imposed by Java applets. Nair's simulation model applets incorporate JSIM, a library of Java classes specifically created for this simulation project. The user interfaces provided for the example applets are simple and self-explanatory. However, animation is interrupted periodically as the applet pauses to connect with the database on the server. Simple, descriptive statistics are available for the simulation run after the user chooses to exit the simulation. Unfortunately, the window displaying the model closes before the statistics appear, so the user is unable to refer back to the model.

SimKit, created by Arnold Buss and Kirk Stork (Buss and Stork, 1996), runs as an applet on the WWW and takes advantage of Java being object-oriented by using the event graph design approach. SimKit models discrete-event simulations and is particularly geared toward military applications. The main simulation facilitating package, Javasim, is designed to allow for expansion in order to accommodate frameworks for various types of simulations. Javasim makes

extensive use of Java interfaces, which add defined behaviors to identified classes without imposing the hierarchical structure of class inheritance. This structure allows a modeler to add customized tools into SimKit without making changes to Javasim. SimKit permits user interaction through a detailed, model entry form. Additionally, SimKit is combined with a Java-based graphing package designed by Leigh Brookshaw, to allow a useful output of statistics and graphs.

Chapter 5: Motivation for Netsim

As seen in Chapter 4, research in the area of WWW-based simulation is developing rapidly as WWW programming tools develop. Still, much of this research provides only limited, WWW-based simulation capabilities. Referencing the contributions and limitations of the current literature, this chapter will discuss how Netsim answers these limitations and provides a positive, unique contribution to WWW-based simulation.

Thorpe helped introduced the idea of Internet-based simulation and, with the development of DIS as a standardized protocol, simulation developers and users began to realize the potential benefits of networked simulation. Unfortunately, DIS is mainly geared toward specific, technical situations and is not connected to the type of widely-accessed, open-topic, network environment presented by the WWW.

By creating simulation models linked to HTML pages, Neilson and Cole and Tooker set the stage for incorporating analytical informational tools to the WWW environment. Their simulation packages allow user interaction, through the WWW browser, between provided models and any relevant WWW sites. However at this stage, Java was not available. As a result, both of their simulation packages require the model and/or the appropriate, browser-compatible software to reside on the user's machine.
Both Singh and Bradley realized the effectiveness of the Java applet as a WWW multimedia tool. By including informational applets in their science tutorials and research publications they enhance the informational value for the user as well as the potential for the user to reproduce reported results more easily. Unfortunately, neither of these implementations take full advantage of the interactive capabilities of Java, particularly in terms of a general WWW-based simulation tool.

By combining a database for handling data with a Java applet for displaying the model to the user, Nair's JSIM package provides more interactivity for the user. Additionally, because the database resides on the server, JSIM realizes many of the potential advantages of WWW simulation. The main drawback to JSIM is the interruption during simulating caused by constant information transfer between the database and the applet.

SimKit is written entirely in Java, eliminating the need for the model to exist on the user's machine and for a connected database. SimKit allows a fair degree of user interaction through a form for entering and editing basic model parameters. Additionally, although it does not provide model animation, SimKit does combine with Java-based graphing packages to provide a number of statistical and graphical interpretations. By using the event-graph approach and utilizing the object-oriented nature of the Java language, SimKit allows for future expansion. However, the

current focus is primarily toward military applications.

Netsim, like SimKit, is written entirely in Java and models discrete-event simulations using the event graph approach. A model created in Netsim appears as an applet on an HTML page and is available over the WWW, taking full advantage of the portability of Java. No special software or code is needed on the user's machine.

Netsim provides a maximum amount of user interaction with the simulation model. A programming interface provides a blank template with text fields for the various parts of a simulation model, such as event name and state variables. Any type of basic simulation model may be entered into this interface by following the format described in 6.3. While knowledge of simulation, particularly including the event graph approach, is useful in designing a model in Netsim, no knowledge of Java or simulation modeling is required to enter or modify the model.

A second interface allows user interaction with running the simulation model. This interface not only provides start, pause, and stop capabilities and data output, but also an animation of the model. Model animation allows viewers a visual demonstration of how the system is operating over time. This is an important feature of Netsim in that users of all education backgrounds can use this WWW-based simulation tool to visually understand the operation of a system or process.

Written as a general simulation package that allows users to define and customize models, Netsim provides more model flexibility than SimKit. The object-oriented structure offered by Java and maintained in Netsim allows easy expansion of the package as well as compatibility with other Java-based tools such as the graphing or analytical tools used by SimKit.

Netsim incorporates all the advantages of WWW simulation presented in Chapter 2 while overcoming all the limitations presented in the current WWW-based simulation literature. Netsim enhances the WWW environment, offering an effective, general simulation tool that provides both model animation and data output. Additionally, Netsim's capabilities for expansion and combination with other Java tools ensure that Netsim can grow with the technology supported by Java and the WWW.

Chapter 6: Netsim Simulation Package

As discussed in Chapter 5, the Netsim simulation package supports discrete-event simulation using event-graph modeling. The current version handles up to six events and six total connections between events and is easily expandable. Consisting of two user interfaces, an input interface for defining and editing the model and an output interface for viewing the model, Netsim features complete model creation and modification capabilities as well as standard simulation, data collection capabilities. Netsim also allows the user to choose the random number seed, the run-length and the output mode (animation and/or data). Netsim runs as a Java applet on any Java-compatible WWW browser or applet viewer. Additionally, a basic user's manual written in HTML accompanies the Netsim package and, through the WWW, can be hypertext linked to the HTML page containing the Netsim applet (Appendix I). The manual explains how to create and interact with a model using Netsim.

Figure 2 outlines the relationship among all classes specifically created for and contained within the Netsim package. The classes of Netsim extend various Java classes as shown in the Netsim documentation and source code (Appendices G and H). Currently none of the Netsim classes

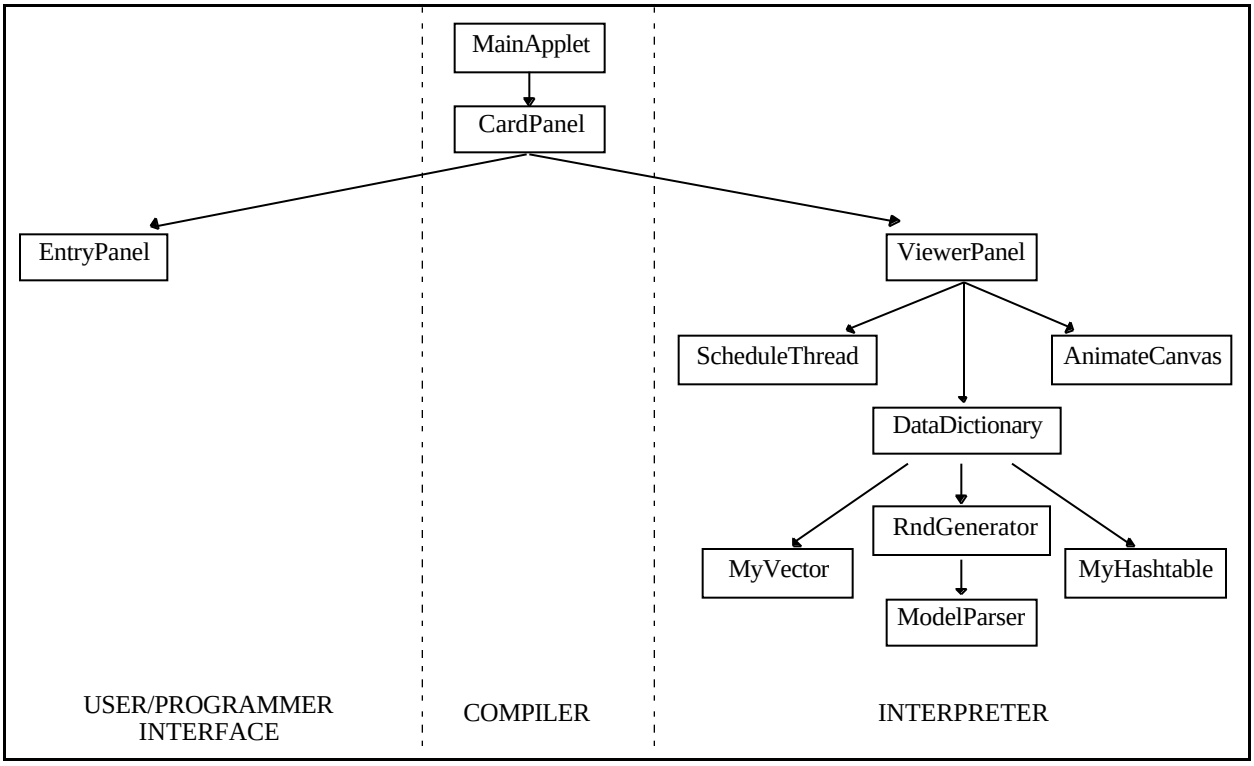

Figure 2: Netsim package internal structure

extends another class within Netsim. Consequently, it should be understood that Figure 2 is not a class hierarchy diagram in the sense that a class in the lower level of the chart extends a class in an upper level. Rather, upper level classes invoke instances of connected, lower level classes and call methods within those classes. The package must be compiled as a group or the classes compiled in succession from the lowest level upward.

6.1 Overview

The Netsim simulation package can be considered from a number of different levels. At the most general level, Netsim consists of three interconnected pieces: a user/ programmer interface, a compiler, and an interpreter as shown in Figure 2.

The user/ programmer interface provides a means for the programmer to introduce a model into the simulation package. This interface may be designed to prompt the programmer for specific information and specific formats. Alternatively, the interface may be as simple as a text window with the expectation that the programmer is familiar with the appropriate format. In the case of Netsim, this interface, shown in Figure 3, resembles a general entry form with text fields for the necessary information. The model user can also use this interface to check and modify model parameters.

Netsim takes advantage of a hypercard type layout format in Java to link the programmer's entry form with the viewing interface on which the interpreter will display the simulation output. Using password restrictions on this interface link, Netsim may be setup so that the programming interface or modifications to the model through it are strictly limited to an authorized programmer. Alternatively, access may be left unrestricted allowing any user to create and edit his/her own model.

In both interfaces Netsim makes extensive use of the java.util.StringTokenizer class for parsing data input. This class is extended by ModelParser and greatly simplifies the compiling process.

The compiler reads the data input into the entry form and parses them into forms usable by the interpreter. Because no special compiler software is needed, the model is instantly executable with a single mouse-click after a being entered into Netsim.

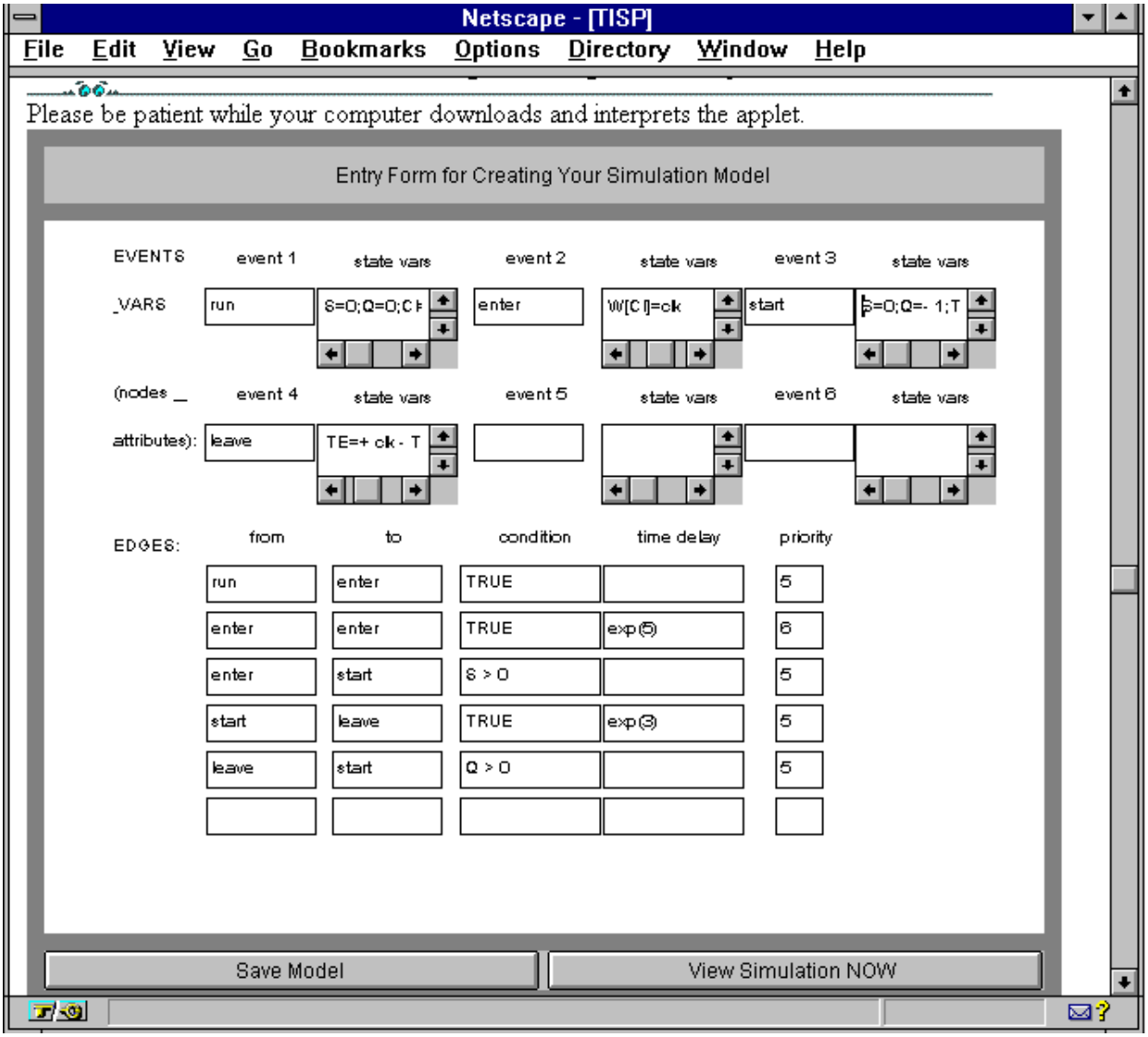

Figure 3: Interface for creating and modifying model

Finally, the interpreter combines the model specifications entered in the entry form with information preprogrammed into the simulation package, such as random variate formulas, to create a dynamic version of the simulation model. The interpreter provides the user with an animation of the model as well as requested data output as shown in Figure 4. Either of the output options may be temporarily canceled, allowing the viewer to get a feel for the general behavior of the system by focusing solely on the animation, or providing the data output at a much higher speed.

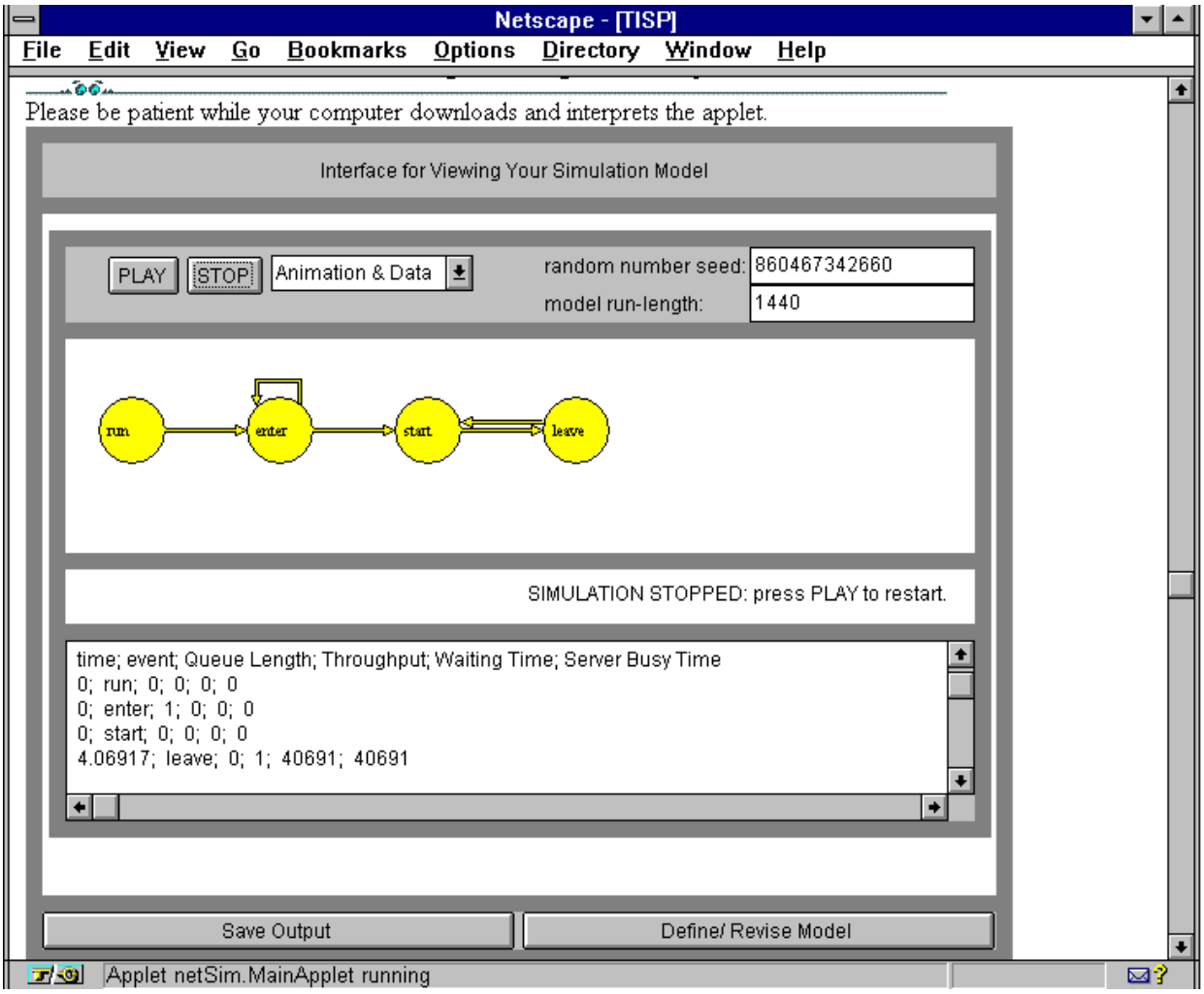

Figure 4: Interface for viewing model animation and data output

6.2 Information Flow

Like Netsim as a whole, the interpreter section of the package can be split into three main sections: the database, the DataDictionary class; the interfaces for the user, the ViewerPanel, AnimationCanvas and EntryPanel classes; and the simulator, the SchedThread class. The database holds the information specific to a given model and supplies these data to the viewing interface as well as to the user/ programmer interface. These interfaces present the model data to the user in a graphical, user-friendly way and enable interaction with the model. Certain user commands on the viewing interface change the status of a simulation run. The run status is passed to the simulator, which then determines the next step in the run. This information is sent to the database to process, resulting in new and revised data for the display on the interfaces. Figure 5 depicts this ongoing cycle. The following subsections discusses each of the three pieces in more detail.

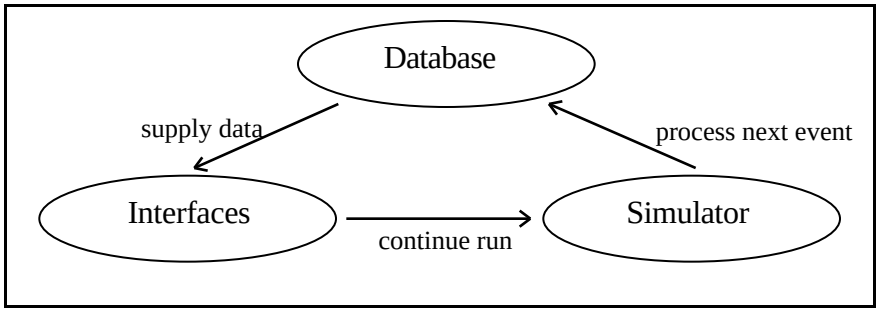

Figure 5: Information flow within Netsim

6.2.1 Database

The database categorizes initial input according to its role in the model and then transfers each type of input into a functional data form that it can manipulate and analyze. The data are stored in hashtables or vectors until needed. Once a simulation run begins, the database takes the next event from the future events list of the simulator, calculates new state variables and random variate values, as necessary, and updates existing database values. Finally, the database updates the future events list and sends requested data to the viewing interface.

6.2.2 Interfaces

The viewing interface displays the simulation model graphically, redrawing the screen as data changes are received from the database. It also displays data output, such as the time and name of the current event. This interface and the user/programmer interface link the user to the heart of the simulation program by presenting model specific information and animating the behavior of the model. Additionally, the viewing interface provides tools such as menus and buttons that allow the user to alter model parameters and control the status of the simulation run. The current run status as well as any changes in the model parameters are passed to the simulator.

6.2.3 Simulator

The simulator accepts run status and model parameter information from the viewing interface in addition to monitoring the simulation clock and the future events list. Using the run status, specified run-length, current clock time and future events list, the simulator determines the next action of the simulation. The simulator sends this information to the database, enabling the database to process the next event and update variable values.

6.3 Model Creation

At any time a user may switch between the user/ programmer interface and the viewing interface by clicking on the lower right button of the screen, which is labeled “View Simulation” or “Define/ Revise Model” depending on the current interface. The save buttons on the lower left side of the screen are currently disabled. Ideally, they would allow the user to name and save either the data output or the model specifications into a directory on the local computer. The data output, as a text file, can then be imported into traditional spreadsheet and analysis packages and is available for further analysis by the user. The model specifications, also as a text file, would act as parameters that, when opened in Netsim, define the saved model. Due to security restrictions, Java applets themselves are currently not able to be saved in this type of manner. While saving and retrieving text from an applet as described is theoretically possible, it was not attempted in this version of Netsim and actual limitations and implementation difficulties are unclear.

To create a model the programmer simply enters the name of each event into the entry form interface, followed by the state variable rules for that event. All names and variables are case-sensitive, should begin with a letter, and should not contain any spaces. The events will appear on the animated model from left to right in the order they are entered on the entry form. The state variables will also be processed in the order they are listed. This may make a difference in the results of a model if one variable references another during the same event.

State variable rules must be separated from the variable name by an equals sign and from another variable equation by a semicolon, (e.g., {1st var. name}={1st var. rule};{2nd var. name}={2nd var. rule}; etc.) Note there is no space between any two of these sections. Each event may contain as many variables and variable rules as desired. A variable rule may do the following, for the example variables Q, TS, T, TE and the reserved variable W[ ]:

- change the existing value by assigning a new integer value,
  (e.g., Q=1 , where Q is the variable name and 1 is the new variable value).
- increase or decrease the existing value by an integer amount,
  (e.g., Q=+ 1 or Q=- 1, with a space left between the operation and the integer).
- increase or decrease the existing value by another variable,
  (e.g., Q=+ TS or Q=- TS, again with a space left between the operation and the value).
- assign the current simulation time to a variable using the reserved word “clk,”
  (e.g., T=clk).
- increase or decrease the existing value by “clk,” the current simulation time,
  (e.g., T=+ clk, again with a space between the operation and the value).
- assign the current “clk” time to the reserved variable array, W[ ], using another variable to index the array,
  (e.g., W[Q]=clk creates an array of clock times indexed by Q).
- combine any of these six operations, except that the array, W[ ], may only contain

"clk" times and should come at the end of the equation,
(e.g., TE=clk - TS or T=+ clk - W[Q]).

Edges may be created between any two consecutive events, or an edge may be self-scheduling (i.e., from an event back to the same event). Netsim currently allows one edge in each direction between two different events, with up to one condition on each edge. These limitations, as well as the current limitation of six event nodes, may be eliminated by extending the current code. However, the interfaces may have to be revised to effectively display on screen the model specifications and the event graphs of a larger model.

The programmer creates an edge by typing in the name of the event where the edge begins in the "from" text box and the event where the edge ends in the "to" text box. These names are case-sensitive and must match the event names defined in the upper section of the event form or the edge will not appear in the model.

Basic conditions may be placed on the edges by using the format shown in Table 1. The variable name may be any variable defined as a state variable in the upper section of the event form. The operator is <, >, or =; the integer value may be any integer. Note a space separates the operator from each of the other terms. The reserved word "TRUE" typed without the quotations in the "condition" text box, makes that edge unconditional.

	var. name	<space>	operator	<space>	integer value
Examples:	Q		<		1
	Q		>		-1
	Q		=		0
			TRUE		

Table 1: Format for edge conditions

Additionally, time delays are added to the edges by typing the appropriate function in the "time delay" text box. Integer-valued parameters of the function are separated by commas and enclosed by parentheses. These time delay functions are case-sensitive and contain no spaces. Netsim currently supports three types of time delays: constant increases, sta(a); uniform random variates, uni(a,b); and exponential random variates, exp(a). The parameters "a" and "b" may be any integers chosen by the user. For example, the delay defined by sta(5) causes constant time increments of 5 time units, uni(3,5) causes a delay uniformly distributed between 3 and 5 time units, and exp(5) delays the upcoming event an exponential amount of time with a mean of 5 time units. Netsim is easily expandable to include other random distributions.

The user alters the priority of an edge by typing the preferred priority, integers 1 to 9, in the "priority" text box. The priority is set by default at 5 with 9 signaling an immediate priority and

1 being a very low priority. This priority scale, while backwards from SIGMA's, seems to be slightly more intuitive, particularly to the occasional user. With Netsim an edge requiring a higher priority should be assigned a higher priority value.

Chapter 7: Analysis of Netsim

7.1 Random Number Generator in Java

The random number stream generated by the class java.util.Random was tested for uniformity and independence using the Chi-square test and two versions of the Runs test. Data are presented in Appendix B while results are summarized in Table 2: Test results of the Java random number generator and discussed below.

For the Chi-square test the [0,1] interval was partitioned into 1,000 even sections, leaving 999 degrees of freedom. Thirty independent runs of 10,000 numbers each were tested. Theoretically, 1.5 of these runs should result in a test value having a p-value less than 0.05. As shown in Table 2: Test results of the Java random number generator, the two largest Chi-square test statistics for the thirty runs have $p < 0.05$ with the third maximum test value being 1073.6 with $p = 0.050$. These results follow the expected distribution and the hypothesis of uniformity is not rejected.

For independence tests thirty independent sets of 10,001 numbers were used. Both tests for independence use a two-tailed Z test. The Runs Up and Down test gave a maximum test statistic of 1.874, with $p = 0.061$. The Runs Above and Below the Mean test resulted in a maximum test statistic of 2.189, $p = 0.029$, and a second largest test statistic of 1.941, $p = 0.051$. Again, out of thirty runs, one to two are expected to result in a p-value less than 0.05 as was the case in the Runs Above and Below the Mean test. For each independence test's set of thirty runs, Figure 6 displays a frequency count of the resulting p-values . As shown by Figure 6, both sets are normally distributed. Additionally, a visual scan of runs of 10 numbers each shows no apparent pattern in the length of runs for either of the above independence tests. Hence, the hypothesis of independence is not rejected.

	Test Statistic:	p-value:
Chi-square:		
Maximum value	1106.7	0.010
Second highest	1078.1	0.041
Third highest	1073.6	0.050
Runs Up and Down:		
Maximum value	1.874	0.061
Runs Above and Below:		
Maximum value	2.189	0.029
Second highest	1.941	0.051

Table 2: Test results of the Java random number generator

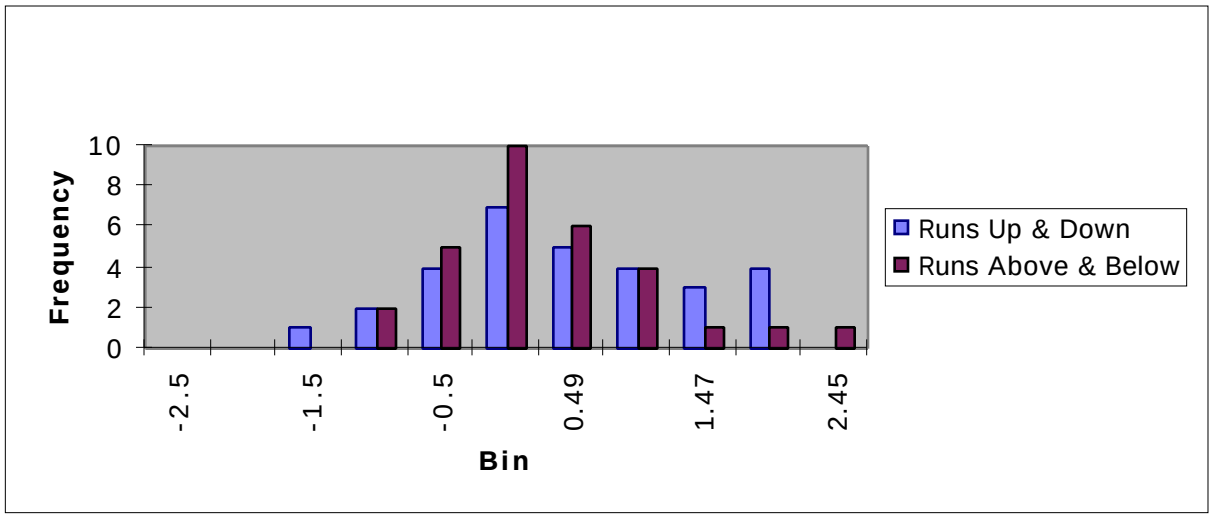

Figure 6: Frequencies of p-values for each set of independence test runs

Due to the results of this section, the random number generator used by the Random class, a standard part of the Java language, was determined to be both uniform and independent.

7.2 Comparison of Netsim with SIGMA

Netsim was compared to SIGMA using the carwash model as presented in Figure 2.6 on page 26 of the SIGMA documentation (Schruben, 1995). This model is a standard G/G/1 queueing model and provides some measure of control in the comparison between the simulation packages. The event graph and state variable rules of the carwash model are given in Figure 7 with the variables defined in Table 3. The time delay for arrivals was exponentially distributed with a mean of 5 time units, while the service time was exponentially distributed with a mean of 3 units.

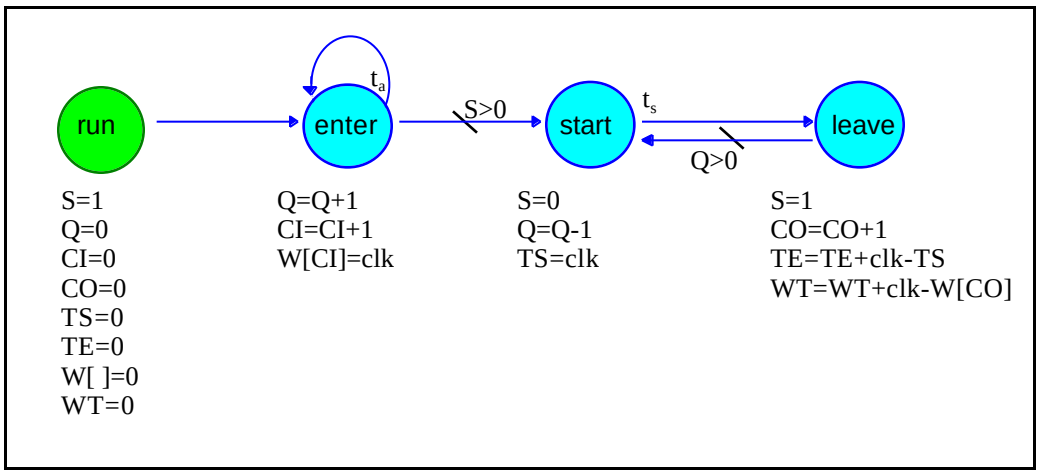

Figure 7: Event graph for carwash model

State variables in model:

S = # of servers available
Q = # of customers in queue
CI = # of customers that have entered system
CO = # of customers that have left system
TS = time customer begins service
TE = total time server is busy throughout run
W[ ] = array tracking the time of each entry into the system
WT = total accumulated waiting time of customers in the system
t_a = interarrival time, exponential with a mean of 5 time units
t_s = service time, exponential with a mean of 3 time units

Variables collected for data comparison:

Q = # of customers left in the queue
CO = system throughput
WT = total waiting time
TE = total server busy time

Table 3: Variable definitions for carwash model

In case of scheduling ties between interarrivals and services, a higher priority was placed on the enter-enter edge than on the start-leave edge. On a scale of 1 to 9 the higher priority is 6 in Netsim while in SIGMA it is 4; this is due to the inverse relationship of the two programs' priority scales. The default priority for both Netsim and SIGMA is 5.

7.3 Random Variate Calculations in NetSim and SIGMA

To calculate an exponential random variate using the inverse transformation technique, one takes the natural log of a uniform random number and multiplies by the negative of the exponential rate. Given a single random number, the exponential random variate calculated by this formula should be the same regardless of differences in software or hardware.

The first column, ARND, of Table 4 consists of a stream of uniform random numbers calculated by SIGMA with the seed 12352. In the second column of Table 4, ln(ARND) is the natural log of ARND as computed by Netsim, or by a standard calculator. The third column depicts TLN, the natural log of ARND as computed by SIGMA. As shown, the values of ln(ARND) and TLN differ slightly. Because of this difference the TLN values are input in Netsim for model comparison between Netsim and SIGMA. The comparison model is described in Section 7.2 and the comparison discussed in Section 7.4. Using the stream of random numbers, ARND, Appendix D supplies a complete table of the exponential random variates, as calculated by SIGMA, for the comparison.

ARND	ln(ARND)	TLN
0.096	-2.343	-2.336
0.754	-0.282	-0.281
0.236	-1.444	-1.439
0.246	-1.402	-1.400
0.740	-0.301	-0.299
0.583	-0.540	-0.538
0.095	-2.354	-2.352
0.336	-1.091	-1.087
0.669	-0.402	-0.401
0.326	-1.121	-1.118
0.017	-4.075	-4.036
0.831	-0.185	-0.185
0.083	-2.489	-2.477
0.034	-3.381	-3.360

Table 4: Natural log calculations

7.4 Comparison Results and Discussion

To accurately compare treatment of the carwash model between Netsim and SIGMA, the natural log values produced by SIGMA (TLN values shown in Table 4, column 3) were input directly into Netsim, bypassing the usual random number generator and natural log calculator. These values were then used by both Netsim and SIGMA in calculating the required exponential random variates. Output at each time unit of activity was compared for the time unit, event, number in queue, accumulated system throughput, accumulated time spent in system (or waiting time), and total server busy time. Appendix E shows this data output from both simulation packages, with the variable names as defined in Section 7.2. As shown, both packages present identical results to at least the second decimal, differing by at most 0.001. This slight difference is non-accumulating during simulation runs and quite possibly results from package differences in byte storage and manipulation.

The future events lists for the two packages matched, as seen in Appendix E, and, when hand traced, corresponded to the theoretical behavior of the model. At each step of the run, state variables and random variates matched between packages and with theoretical values.

In addition, the future events list generated by Netsim was traced for numerous variations, both stochastic and deterministic, of the carwash model. State variables at each simulation step, as well as overall model behavior were examined in each case and, for deterministic models, were compared to those from identical models run in Sigma. In all cases the models performed as expected by discrete-event simulation theory.

7.5 Netsim Limitations

7.5.1 Conversion of Clock Values

In the current version of Netsim in order to maintain a compatible format among variable values in the database, "clk" must be an integer. However, the simulation clock time is taken from uniform(0,1) random variates calculated by the java.util.Random.nextDouble() method in Java. To solve this discrepancy, the clock time, when used in the variable database, is first converted to an integer. This is done by multiplying by 10,000 and truncating the remaining decimal portion. The new value is equivalent to "clk." Time-valued variables are easily rescaled by dividing by 10,000.

In Table 5, a sample of rescaled data output using the 10,000 multiplier is compared to a run with the same seed and using a 1,000,000 multiplier. As shown, the 10,000 multiplier reduces the number of significant digits in the data output to at most three, but does not otherwise affect the accuracy of the output. The number of significant digits can be increased by increasing the multiplier. However, this causes variable values to quickly exceed the predefined integer size limit. Potential solutions to this conversion problem for future versions of Netsim are discussed in Chapter 8.

multiplier	time	event	Q	CO	WT	TE
10,000	14433.98	leave	0	266	2006.2937	82.81475
1,000,000	14433.98	leave	0	266	2006.294220	82.8147698

Table 5: Sample of simulation run using different clock multipliers

7.5.2 Thread Synchronicity

The current version of Netsim contains the following, surface level, oddities caused by lack of proper synchronicity among the threads. These inconsistencies affect viewing the animation, but do not affect the simulation process or accuracy of the data output. In particular, in order to rerun the simulation the user must actually click on the stop button before clicking the play button, even if the simulation has finished naturally. Additionally, when viewing a Netsim model on a WWW browser, pausing the animation and temporarily activating a different window on the computer screen may cause the simulation run to resume. Finally, the pause button is not reliable on machines, such as the Sun, that use multithreaded processors. An effective fix to these problems will involve a thorough understanding of multithreading capabilities and behaviors within the Java language (Lemay and Perkins, 1996).

Chapter 8: Conclusions

8.1 Benefits of WWW Simulation

In computer applications involving long-term use from a single machine, a traditional, non-Internet-based simulation package may be appropriate. Such a package is often developed to handle specific, complex problems and can be customized to work optimally with a particular computer platform. Additionally, it can generally be expected to perform with a great deal of reliability as software availability and access times depend only on one computer, not on the Internet network.

The expanding popularity of the WWW, however, suggests a growing number of situations efficiently modeled through WWW-based simulation. Companies may find WWW-based simulation tools helpful in monitoring business performance and in demonstrating the features of a new product. Other potential applications include educational tutorials, skill training and role playing games. WWW-based simulation presents these and other applications with many advantages over non-Internet-based simulation. Several important advantages are summarized in the following paragraphs.

WWW-based simulation, particularly when created using the Java programming language, can be made widely available. Such simulation models are accessible from any Internet-connected computer through a number of computer platforms and without recompilation. Also, because the Internet is usually accessible twenty-four hours a day, simulation modeling is possible during evening and weekend hours as well as normal business hours.

WWW-based simulation models can be protected from inadvertent or unauthorized modifications by the user. This ensures distribution of identical models to all users and minimizes file maintenance issues for the user. In addition, providers of WWW-based simulation models can easily control access to the site by imposing password and time limit restrictions on either the model or the entire site.

WWW-based simulation models run from a single WWW site enabling both reliable version control and frequent model modifications. Additionally, the most up-to-date versions of the model are instantly available through the server to all authorized Internet users, regardless of physical location.

Finally, WWW-simulation models created with Java require only a Java-compatible WWW browser for viewing. Such browsers are easy to install and use, allowing users to access models quickly. By providing the browser as a simple access tool, along with many tools for

communication, interaction and data access, the WWW environment can enhance the informational value of the simulation model.

8.2 Contributions of Netsim

The current literature for WWW-based simulation motivates the development of Netsim, a general, discrete-event, WWW-simulation package. Programmed entirely in the WWW-compatible Java language, Netsim offers all the previously discussed advantages of WWW-based simulation. Additionally, Netsim allows users full interactive capabilities and provides both model animation and data output. Users may create, modify and interact with a simulation model, receiving requested data output. Meanwhile, the animation capabilities of Netsim present a visual representation of the behavior of the system. By providing a means for general simulation modeling in connection with these advantages and capabilities, Netsim enables most, if not all, of the users in the WWW environment to take advantage of simulation modeling as an analytical, informative tool.

Netsim takes advantages of the drawing and string parsing portions of Java to increase cross-platform compatibility and allow users maximum interaction capabilities. The Abstract Window Toolkit in Java aids the creation of graphical user interfaces in Netsim and allows the interfaces to be rendered according to the platform-dependent information in the user's WWW browser, ensuring maximum graphical compatibility with that platform. When users enter and modify models, Netsim extends the Java StringTokenizer class to parse the entered data from string format into the proper format for the databases.

Additionally, Netsim uses threads and flexible database structures to minimize download time and unnecessary memory usage within the applet. Threads are used in classes, such as the interface creation classes, that define numerous variables, graphical components, and/or data structures. The threads decrease package startup time by allowing classes to define and instantiate themselves simultaneously. Netsim uses databases composed of hashtables and vectors that expand themselves as necessary instead of being initially defined with unchanging sizes. As a result, the amount of memory required by the Netsim applet at a given time depends on the number of events, edges, and variables within the current model. Additionally, this composition of databases contributes to the ability of Netsim to model varied types and sizes of models.

At this time, reasonable size limits on Java applets, particularly in terms of download time and browser or client platform memory allocation, are unclear; current documentation does not specify a suggested or originally targeted size limit. However, because of WWW security issues, applet memory allocation is made at run-time and, consequently, is dependent on the client software and hardware (Gosling, 1996). As seen with Netsim, applets of a substantial nature can

be utilized without requiring extreme download times or amounts of free memory on the client platform.

As demonstrated in Chapter 7 with the carwash model, Netsim codes and processes simple, discrete event simulation models accurately. Whether or not the models themselves are valid for their respective systems is up to the modeler and programmer, as is the case with any simulation software. Because Java applets such as Netsim are entirely downloaded onto the local computer before interpretation into a platform-specific, executable program, download times for Netsim are proportional to a given computer's Internet connection and to the level of current activity on that part of the Internet.

8.3 Programming in Java

As an object-oriented programming language designed for simplicity and robustness, Java is, in fact, relatively simple and well-behaved. The Java language contains a set of class libraries that provide data structures and handle common utility functions. Also, the descriptive names allowed in Java simplify documentation and mental bookkeeping by helping the programmer and other readers follow method calls within a program. Furthermore, much Java coding resembles that of established object-oriented languages like C++. However, unlike C++, Java does not use reference pointers that can cause nearly undetectable memory problems. Additionally, self-initiated threads in Java manage applet memory allocation and garbage control throughout the life of an applet. These features allow more focus on programming and allow program experimentation without disastrous results such as system crashes.

For a novice Java programmer, more difficult coding involves forcing the graphical components to refresh at appropriate times. Within an applet, Java uses a self-initiated thread to control painting and repainting onto the screen. While this is beneficial in many applets, those relying on graphical display techniques such as animation require particular refresh rates. Because the involved Java-defined methods create a required graphics object that can not be explicitly instantiated, calling these methods from user-defined methods that describe repainting behaviors can be tricky. An inelegant solution can often be obtained through less than true object-oriented programming where most or all behaviors involved in painting and repainting are included into the Java-defined methods, not into the multiple user-defined methods suggested by the logic of the particular program.

Netsim was created using the Java Development Kit (JDK), version 1.0.2. The Macintosh version of the JDK includes an applet viewer with several user-interface bugs that slow development. However, on all platforms, the JDK compiler error information is helpful. Also, a number of graphical development environments for Java were marketed during the development of Netsim; by allowing drag and drop programming techniques, such software should noticeably

speed Java applet development, particularly for infrequent programmers. There are small typographical errors or confusing issues with a number of the examples in the cited Java resource books that can cause minor setbacks in the learning process if one is not actively testing the examples while reading. However, during development of Netsim, the number of Java programming books has expanded greatly, hopefully providing a more exhaustive and comprehensive resource library. Additionally, Java maintains a web site with Application Programming Interface (API) documentation, white papers and information about current product releases (Sun, 1997).

8.4 Future Work

The current version of Netsim contains a small number of limitations as discussed in Chapter 7, Section 7.5. These are programming issues which can be remedied with more complex programming techniques than undertaken for the creation of Netsim. In particular, the creation of a special database could prevent the loss of significant digits caused by integer translation of simulation clock time. Such a database would have to recognize and be devoted specifically to time-valued variables, managing those variables separately from other variables. Viewing inconsistencies discussed in Section 7.5.2 are remedied by carefully synchronizing the threads within the package. However, because some of these threads reference methods within each other, proper solution of this problem will necessitate solid knowledge of multithreading. None of the discussed limitations directly affect the simulation process and both involve programming techniques outside the scope of the current research.

Because Netsim was specifically created as a general simulation package, expansion, in most cases, simply involves extension through additional classes and/or methods. The object-oriented structure of Netsim greatly facilitates this process. Future additions should include classes for handling more advanced simulation models, support for a wider range of random variates, and additional user interfaces. Databases holding edge information should be expanded to allow parameter passing, enabling complex modeling through simple event graphs. Additionally, user understanding would be enhanced by replacing event graphs with system-appropriate icons and designs and by including context-sensitive help within the applet. For users concerned with replicating or presenting the data output, future modifications include enabling the current “save” buttons as well as adding graphing and statistical analysis capabilities.

Use of the Java language is expanding rapidly, especially in the creation of educational software. Consequently, some of the potential enhancements mentioned for Netsim may be made possible by linking with other existing Java-based classes, in their current form or with minor modifications.

The information potential through the WWW is growing rapidly, encouraging more and more

people of all backgrounds and interests to gain regular, frequent access to the WWW. Additionally, the increase of standardized HTML formats, multimedia browser plug-ins, and scripting and programming languages are allowing WWW site providers to create sites that are more interesting, visual, and user-friendly. As the popularity of the WWW for everyday transfer of information and ideas expands, WWW-based simulation may become an extremely beneficial and desired addition to the common WWW environment. For example, by providing a high level of user interactivity through simple interfaces, Netsim encourages users of all levels of understanding to interact with the model. Additionally, through animation and data output, Netsim contributes a graphic as well as analytic representation of a system process. This enables users to understand the system both visually and statistically. By accessing the Netsim simulation models through the WWW users are able to ensure a common reference and to communicate easily with each other, building on understandings and ideas.

References and Bibliography

Afergan, Micheal M. (1996) **Java™: Quick Reference**. Que, Indianapolis, IN.

Aitken, Gary. (1996) *Moving from C++ to Java*, Dr. Dobb's Journal. March, 21(3): 52-56.

Apple Computer, Inc. (1997) **Apple Computer's web site for OpenDoc developers: About OpenDoc**. http://www.opendoc.apple.com/press/od-cdog.html.

Apple Computer, Inc. (1996) **OpenDoc Development Framework Overview**. http://www.devtools.apple.com/odf/overview.html.

Banks, Jerry. (1996) *Interpreting Simulation Software Checklists*, OR/MS Today. June - Special International Issue, 74-78.

Banks, Jerry, John S. Carson, II and Barry L. Nelson. (1996) **Discrete-Event System Simulation: second edition**. Prentice Hall, New Jersey.

Bradley, Gordon H. (1996) *Dynamic and interactive research publications using Java*, http://dubhe.cc.nps.navy.mil/~gbradley/. February.

Buss, Arnold H. (1996) *Modeling with Event Graphs*, Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11.

Buss, Arnold H. and Kirk A. Stork. (1996) *Discrete Event Simulation on the World Wide Web Using Java*, Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11. http://131.120.142.115/~stork/simkit_home.html.

Cole, Rodney and Scott Tooker. (1996) *Physics To Go: Web-based Tutorials For CoLoS Physics Simulations*, Technology-Based Re-Engineering. Engineering Education. Proceeding of the Frontiers on Education FIE'96 26th Annual Conference. 2: 681-683.

December, John. (1995) **Presenting Java: An Introduction to Java and HotJava**. Sams.net, Indianapolis, IN.

Fishwick, Paul A. (1996) *Web-based Simulation: Some Personal Observations*, Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11. http://www1.cise.ufl.edu/~fishwick/websim.html.

Freeman, Adam and Darrel Ince. (1996) **Active Java: Object-Oriented Programming for the World Wide Web**. Addison-Wesley, Harlow, England.

Fullford, Deb. (1996) *Distributed Interactive Simulation: It's Past, Present and Future*, Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11.

Gosling, James and Henry McGilton. (1996) ***The Java Language Environment: A White Paper***, http://www.javasoft.com/docs/language_environment/. May.

Joines, Jeffery A., and Stephen D. Roberts. (1996) *Design of Object-Oriented Simulation in C++*, Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11.

Jones, Christopher V. (1996) *Java and OR/MS*, INFORMS CSTS Newsletter. Spring, 17(1): 3-11.

Law, Averill M. and W. David Kelton. (1991) **Simulation Modeling & Analysis: second edition**. McGraw-Hill, New York, New York.

Lemay, Laura and Charles L. Perkins. (1996) **Teach Yourself Java™ in 21 days**. Sams.net, Indianapolis, IN.

Nagaratnam, Nataraj, Brian Maso, and Arvind Srinivasan. (1996) **Java™ Networking and AWT API Superbible**. Waite Group Press, Carte Madera, CA.

Nair, Rajesh S., John A. Miller, Zhiwei Zhang. (1996) *Java-Based Query Driven Simulation Environment*, Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11. http://www.cs.uga.edu/~rajesh/JSIM/jsimdistribution/docs/API_users_guide.html.

Neilson, Irene, Ruth Thomas, Calum Smeaton, Alan Slater, and Gopal Chand. (1996) *Education 2000: Implications of W3 Technology*, Computers and Education. 26(1-3):113-122.

Netscape Communications Corporation. (1997) **Netscape Navigator**. Mountain View, CA. http://home.netscape.com/.

Pritsker, A. Alan B. (1986) **Introduction to Simulation and SLAM II: third edition**. John Wiley & Sons, New York.

Schruben, Lee W. (1995) **Graphical Simulation Modeling and Analysis: Using SIGMA for Windows**. Boyd & Fraser, Danvers, MA.

Schulzrinne, Henning. (1996) *World Wide Web: Whence, Whither, What Next?*, IEEE Network. March/April, 10(2): 10-17.

Schriber, Thomas J. & Daniel T. Brunner. (1996) *Inside Simulation Software: How It Works and Why It Matters,* Proceedings of the 1996 Winter Simulation Conference. ed.: J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain. Coronado, California. December 8-11.

Singh, T., M. Zhu, U. Thakker, and U. Ravaioli. (1996) *Impact of World Wide Web, Java, and Virtual Environments on Education in Computational Science and Engineering*, Technology-Based Re-Engineering. Engineering Education. Proceeding of the Frontiers on Education FIE'96 26th Annual Conference. 3: 1007-1010.

Sun Microsystems, Inc. (1997) **The Source For Java: JavaSoft Home Page**. http://java.sun.com/.

Tooker, Scott and Rodney Cole. (1996) *Using OpenDoc To Create Low-Cost Physics Simulation Tools For Secondary and Higher Education*, Technology-Based Re-Engineering. Engineering Education. Proceeding of the Frontiers on Education FIE'96 26th Annual

Conference. 2: 688-689.
Zar, Jerrold H. (1984) **Biostatistical Analysis: second edition**. Prentice Hall, New Jersey.

Appendices

Appendix A: Copyrights/ Disclaimers

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Tamie Lynne Veith and the Netsim simulation package are independent of Sun Microsystems, Inc.

SLAM II (Simulation Language for Alternative Modeling) is a registered trademark of Pritsker & Associates, Inc.

SIGMA (Simulation Graphical Modeling and Analysis) is a trademark of Systems Design, Inc.

Appendix B: Check of Java Random Number Generator:

Data Output

Uniformity test:		Chi-square	
degrees of freedom:		999	
RNG Seed:	# of numbers:	test value:	
12345	10000	**968.5**	
12346	10000	**1073.6**	
12347	10000	**989.7**	
12348	10000	**984.4**	
12349	10000	**975.8**	
12350	10000	**1015.7**	
12351	10000	**1035.4**	
12352	10000	**976.8**	
12353	10000	**1054.2**	
12354	10000	**971.9**	
12355	10000	**1006.6**	
12356	10000	**1034.8**	
12357	10000	**1023.0**	
12358	10000	**1036.7**	
12359	10000	**986.4**	
12360	10000	**1106.7**	
12361	10000	**1078.1**	
12362	10000	**963.8**	
12363	10000	**1054.4**	
12364	10000	**1014.8**	
12365	10000	**1024.7**	
12366	10000	**1025.6**	
12367	10000	**1049.0**	
12368	10000	**1000.2**	
12369	10000	**986.7**	
12370	10000	**1049.6**	
12371	10000	**966.0**	
12372	10000	**985.0**	
12373	10000	**894.3**	
12374	10000	**1003.5**	
Max (Abs):		**1106.7**	
p-value:		**0.010**	

Independence tests:		Runs Up & Down		Runs Above & Below Mean of 0.5			
RNG Seed:	# of numbers:	# of runs:	**test value:**	# above mean:	# below mean:	# of runs:	**test value:**
12345	10001	6610	**-1.352**	4952	5048	4960	**-0.791**
12346	10001	6706	**0.925**	4919	5081	4980	**-0.374**
12347	10001	6669	**0.047**	4958	5042	4965	**-0.693**
12348	10001	6746	**1.874**	4960	5040	4983	**-0.334**
12349	10001	6718	**1.210**	4944	5056	5026	**0.533**
12350	10001	6608	**-1.399**	4940	5060	4986	**-0.266**
12351	10001	6713	**1.091**	5057	4943	4983	**-0.327**
12352	10001	6641	**-0.617**	5030	4970	5045	**0.904**
12353	10001	6641	**-0.617**	4942	5058	4955	**-0.887**
12354	10001	6673	**0.142**	4963	5037	4948	**-1.035**
12355	10001	6682	**0.356**	4983	5017	5012	**0.241**
12356	10001	6671	**0.095**	4985	5015	5097	**1.941**
12357	10001	6669	**0.047**	4965	5035	4993	**-0.135**
12358	10001	6666	**-0.024**	5005	4995	5004	**0.080**
12359	10001	6658	**-0.213**	5000	5000	5016	**0.320**
12360	10001	6637	**-0.712**	4859	5141	5105	**2.182**
12361	10001	6658	**-0.213**	5028	4972	5029	**0.583**
12362	10001	6742	**1.779**	5020	4980	5018	**0.362**
12363	10001	6746	**1.874**	5002	4998	5026	**0.520**
12364	10001	6649	**-0.427**	4877	5123	4977	**-0.400**
12365	10001	6700	**0.783**	5001	4999	4990	**-0.200**
12366	10001	6667	**0.000**	5036	4964	4992	**-0.155**
12367	10001	6653	**-0.332**	5030	4970	5011	**0.224**
12368	10001	6736	**1.637**	4867	5133	4948	**-0.970**
12369	10001	6705	**0.901**	4945	5055	4997	**-0.048**
12370	10001	6723	**1.328**	5002	4998	5003	**0.060**
12371	10001	6604	**-1.494**	4968	5032	4978	**-0.436**
12372	10001	6708	**0.972**	5001	4999	4930	**-1.400**
12373	10001	6657	**-0.237**	5037	4963	5055	**1.106**
12374	10001	6635	**-0.759**	4936	5064	4967	**-0.644**
Max (Abs):			**1.874**				**2.1817**
p-value:			**0.061**				**0.0290**

Appendix C: Check of Java Random Number Generator: Program

```
1   Random Number Test HTML Document
2
3   <HTML>
4   <HEAD>
5   <TITLE>Random Number Test</TITLE>
6   </HEAD>
7
8   <BODY>
9   <APPLET CODE = "RndNum.class" WIDTH =100 HEIGHT =250>
10  Random Number Test</APPLET>
11  </BODY>
12  </HTML>
```

```
1   import java.awt.*;
2   import java.lang.*;
3   import java.util.*;
4
5   /**
6         File name: RndNum.java
7         Class: RndNum extends java.applet.Applet
8
9                   To test the randomness of Double random numbers
10                  generated by the java.util.Random class.
11  */
12
13  public class RndNum extends java.applet.Applet{
14          double aRnd =0, aRnd2 = 0;
15
16          public void chiTest(long newSeed) {
17                  Random r = new Random(newSeed);
18                  int[] intervals = new int[1000];
19                  double thisValue = 0;
20                  double lstSq = 0;
21                  double totalLstSq = 0;
22                  for(int i = 0; i < 9999; i++) {          //10,000 total observations
23                  aRnd= r.nextDouble();
24          Double aRndDbl=new Double(aRnd*1000);
25          Integer whichBin=new Integer((aRndDbl).intValue());
26                  intervals[whichBin.intValue()]++;  // always falls in interval below
27                  }
28                  //System.out.println("intervals:");
29                  for(int i = 0; i < 999; i++) {
30                          //System.out.print(intervals[i]+"; ");
31                          //System.out.print("(o-e)^2/e = ");
32                          thisValue = intervals[i];
33                          lstSq =(((thisValue-10)*(thisValue-10))/10);
34                          //System.out.println(lstSq);
35                          totalLstSq += lstSq;
36                  }
37                  System.out.println("value to test: "+totalLstSq);
38          }
39
40          public void runsTest(long newSeed) {
```

```
41          Random r = new Random(newSeed);
42          double upCount = 0, downCount=0;
43          double[] runlength;
44          double aboveMean = 0, belowMean=0;
45          int run = 0, runM=0;
46          aRnd= r.nextDouble();
47          boolean switchUp = true;
48          boolean switchM = true;
49          for(int i = 0; i < 10000; i++) {
50                  aRnd2=r.nextDouble();
51                  //System.out.println("aRnd: "+aRnd+" aRnd2: "+aRnd2);
52                  //test general run lengths
53                  if (aRnd<aRnd2) {
54                          if (switchUp) {
55                                  run ++;
56                                  switchUp = false;
57                          }
58                          upCount ++;
59                  } else if (aRnd>=aRnd2) {
60                          if (!switchUp) {
61                                  run ++;
62                                  switchUp = true;
63                          }
64                          downCount ++;
65                  }
66                  //test runs about mean
67                  if (aRnd<0.5) {
68                          if (switchM) {
69                                  runM ++;
70                                  switchM = false;
71                          }
72                          belowMean++;
73                  } else if (aRnd>=0.5) {
74                          if (!switchM) {
75                                  runM ++;
76                                  switchM = true;
77                          }
78                          aboveMean++;
79                  }
80                  aRnd=aRnd2;
```

```
81                      }
82                      //System.out.print("total runs {general}: "+run+" upCount: ");
83                      //System.out.println(upCount+" downCount: "+downCount);
84                      //System.out.print("total runs {mean}: "+runM+" aboveMean: ");
85                      //System.out.println(aboveMean+" belowMean: "+belowMean);
86                      System.out.print("runs: "+run+" aboveMean: "+aboveMean);
87                      System.out.println(" belowMean: "+belowMean+" runsM: "+runM);
88              }
89
90              public void expTest(long newSeed) {
91                      Random r = new Random(newSeed);
92                      double tff = 2.302585;
93                      double aLn=0,aLog=0;
94                      double totalRnd=0,totalLn=0,totalLog=0;
95                      double avgRnd=0,avgLn=0,avgLog=0;
96                      for(int i = 0; i < 10000; i++) {
97                              aRnd =r.nextDouble();
98                              totalRnd += aRnd;
99                              aLn = Math.log(aRnd);
100                             aLog = (Math.log(aRnd)/tff);
101                             totalLn += Math.log(aRnd);
102                             totalLog  += (Math.log(aRnd)/tff);
103                             //System.out.print("aRnd = "+ aRnd+ "; aLog = "+ aLog);
104                             //System.out.println("; aLn = "+ aLn);
105                     }
106                     avgRnd = totalRnd/10000;
107                     avgLog = totalLog/10000;
108                     avgLn = totalLn/10000;
109                     System.out.print("totalRnd = "+ totalRnd+ "; totalLog = ");
110                     System.out.println(totalLog+"; totalLn = "+ totalLn);
111                     System.out.print("meanRnd = "+ avgRnd+ "; meanLog = ");
112                     System.out.println(avgLog+"; meanLn = "+avgLn);
113             }
114
115             public void start() {
116                     RndNum rd= new RndNum();
117                     for(long mySeed = 12345; mySeed < 12375; mySeed++) {
118                             //System.out.println("RND gen for doubles with seed: " +
119     mySeed);
120                             //rd.chiTest(mySeed);
```

```
121                         //rd.runsTest(mySeed);
122                         rd.expTest(mySeed);
123                     }
124                     stop();
125             }
126     }//endof RndNum class
127
```

Appendix D: SIGMA Random Variate Calculations: Data Output

RNG seed:				
	12352			
variables:				
	ARND =	uniform random variate generated by SIGMA		
	TLN =	ln(ARND)		
	TA =	-5*TLN		
	TSER =	-3*TLN		
Count	ARND	TLN	TA	TSER
1	0.096	-2.336	11.682	7.009
2	0.754	-0.281	1.406	0.844
3	0.236	-1.439	7.199	4.319
4	0.246	-1.4	7.003	4.202
5	0.74	-0.299	1.499	0.899
6	0.583	-0.538	2.69	1.614
7	0.095	-2.352	11.764	7.058
8	0.336	-1.087	5.438	3.263
9	0.669	-0.401	2.008	1.204
10	0.326	-1.118	5.594	3.356
11	0.017	-4.036	20.181	12.109
12	0.831	-0.185	0.925	0.555
13	0.083	-2.477	12.387	7.432
14	0.034	-3.36	16.803	10.082
15	0.358	-1.026	5.134	3.08
16	0.856	-0.154	0.772	0.463
17	0.037	-3.291	16.458	9.874
18	0.087	-2.43	12.152	7.291
19	0.95	-0.051	0.255	0.153
20	0.635	-0.453	2.268	1.36
21	0.786	-0.239	1.199	0.719
22	0.016	-4.101	20.508	12.305
23	0.039	-3.231	16.158	9.694
24	0.751	-0.285	1.427	0.856

25	0.853	-0.158	0.792	0.475
26	0.624	-0.471	2.355	1.413
27	0.95	-0.05	0.254	0.152
28	0.207	-1.574	7.874	4.724
29	0.424	-0.857	4.289	2.573
30	0.106	-2.242	11.211	6.726
31	0.282	-1.262	6.313	3.788
32	0.67	-0.4	2	1.2
33	0.819	-0.199	0.996	0.597
34	0.281	-1.267	6.335	3.801
35	0.083	-2.483	12.418	7.45
36	0.349	-1.052	5.261	3.156
37	0.351	-1.045	5.226	3.135
38	0.182	-1.701	8.509	5.105
39	0.293	-1.224	6.124	3.674
40	0.663	-0.41	2.052	1.231
41	0.424	-0.857	4.286	2.571
42	0.461	-0.774	3.87	2.322
43	0.301	-1.199	5.998	3.599
44	0.451	-0.795	3.978	2.387
45	0.557	-0.584	2.922	1.753
46	0.816	-0.203	1.016	0.609
47	0.952	-0.049	0.245	0.147
48	0.902	-0.102	0.513	0.307
49	0.513	-0.666	3.332	1.999
50	0.553	-0.591	2.955	1.773
51	0.743	-0.296	1.481	0.889
52	0.942	-0.058	0.294	0.176
53	0.989	-0.01	0.051	0.03
54	0.649	-0.431	2.159	1.295
55	0.641	-0.444	2.223	1.334
56	0.696	-0.361	1.805	1.083
57	0.462	-0.77	3.853	2.311
58	0.85	-0.161	0.809	0.485

59	0.538	-0.619	3.095	1.857
60	0.09	-2.401	12.007	7.204
61	0.274	-1.29	6.454	3.872
62	0.866	-0.142	0.714	0.428
63	0.238	-1.434	7.17	4.302
64	0.694	-0.364	1.82	1.092
65	0.003	-5.693	28.467	17.08
66	0.6	-0.51	2.554	1.532
67	0.416	-0.874	4.373	2.624
68	0.732	-0.31	1.554	0.932
69	0.784	-0.243	1.216	0.729
70	0.948	-0.052	0.263	0.158
71	0.734	-0.309	1.545	0.927
72	0.923	-0.079	0.396	0.237
73	0.514	-0.664	3.323	1.994
74	0.473	-0.748	3.742	2.245
75	0.439	-0.821	4.107	2.464
76	0.834	-0.18	0.902	0.541
77	0.984	-0.015	0.078	0.047
78	0.637	-0.45	2.252	1.351

Appendix E: Carwash Model: SIGMA & NetSim Data Output

Carwash model run in SIGMA:					
Uses SIGMA-generated random number stream and TLN values (seed=12352).					
Time	Event	Q	CO	WT	TE
0	run	0	0	0	0
0	enter	1	0	0	0
0	start	0	0	0	0
0.844	leave	0	1	0.844	0.844
11.682	enter	1	1	0.844	0.844
11.682	start	0	1	0.844	0.844
15.884	leave	0	2	5.046	5.046
18.881	enter	1	2	5.046	5.046
18.881	start	0	2	5.046	5.046
20.381	enter	1	2	5.046	5.046
20.496	leave	1	3	6.66	6.66
20.496	start	0	3	6.66	6.66
23.759	leave	0	4	10.038	9.923
32.145	enter	1	4	10.038	9.923
32.145	start	0	4	10.038	9.923
34.153	enter	1	4	10.038	9.923
35.502	leave	1	5	13.395	13.28
35.502	start	0	5	13.395	13.28
36.057	leave	0	6	15.298	13.835
54.335	enter	1	6	15.298	13.835
54.335	start	0	6	15.298	13.835
64.417	leave	0	7	25.38	23.917
66.723	enter	1	7	25.38	23.917
66.723	start	0	7	25.38	23.917
67.186	leave	0	8	25.844	24.381
71.857	enter	1	8	25.844	24.381
71.857	start	0	8	25.844	24.381
79.148	leave	0	9	33.135	31.672
88.315	enter	1	9	33.135	31.672
88.315	start	0	9	33.135	31.672

88.571	enter	1	9	33.135	31.672
89.676	leave	1	10	34.496	33.033
89.676	start	0	10	34.496	33.033
89.77	enter	1	10	34.496	33.033
101.982	leave	1	11	47.907	45.338
101.982	start	0	11	47.907	45.338
102.838	leave	0	12	60.975	46.195
105.929	enter	1	12	60.975	46.195
105.929	start	0	12	60.975	46.195
106.721	enter	1	12	60.975	46.195
106.976	enter	2	12	60.975	46.195
107.342	leave	2	13	62.388	47.609
107.342	start	1	13	62.388	47.609
109.916	leave	1	14	65.583	50.182
109.916	start	0	14	65.583	50.182
114.85	enter	1	14	65.583	50.182
116.642	leave	1	15	75.25	56.909
116.642	start	0	15	75.25	56.909
117.843	leave	0	16	78.242	58.11
121.164	enter	1	16	78.242	58.11
121.164	start	0	16	78.242	58.11
122.16	enter	1	16	78.242	58.11
124.965	leave	1	17	82.043	61.911
124.965	start	0	17	82.043	61.911
128.121	leave	0	18	88.005	65.067
134.578	enter	1	18	88.005	65.067
134.578	start	0	18	88.005	65.067
139.684	leave	0	19	93.111	70.173
139.804	enter	1	19	93.111	70.173
139.804	start	0	19	93.111	70.173
141.036	leave	0	20	94.343	71.405
145.929	enter	1	20	94.343	71.405
145.929	start	0	20	94.343	71.405
148.251	leave	0	21	96.665	73.727

150.215	enter	1	21	96.665	73.727
150.215	start	0	21	96.665	73.727
152.602	leave	0	22	99.052	76.114
156.214	enter	1	22	99.052	76.114
156.214	start	0	22	99.052	76.114
156.824	leave	0	23	99.662	76.724
159.136	enter	1	23	99.662	76.724
159.136	start	0	23	99.662	76.724
159.382	enter	1	23	99.662	76.724
159.444	leave	1	24	99.97	77.032
159.444	start	0	24	99.97	77.032
161.218	leave	0	25	101.805	78.806
162.714	enter	1	25	101.805	78.806
162.714	start	0	25	101.805	78.806
162.891	leave	0	26	101.982	78.982
164.196	enter	1	26	101.982	78.982
164.196	start	0	26	101.982	78.982
164.248	enter	1	26	101.982	78.982
165.492	leave	1	27	103.278	80.278
165.492	start	0	27	103.278	80.278
166.471	enter	1	27	103.278	80.278
166.575	leave	1	28	105.605	81.361
166.575	start	0	28	105.605	81.361
167.061	leave	0	29	106.196	81.847
170.324	enter	1	29	106.196	81.847
170.324	start	0	29	106.196	81.847
173.419	enter	1	29	106.196	81.847
177.529	leave	1	30	113.4	89.052
177.529	start	0	30	113.4	89.052
177.958	leave	0	31	117.939	89.481
179.874	enter	1	31	117.939	89.481
179.874	start	0	31	117.939	89.481
180.966	leave	0	32	119.031	90.573
187.045	enter	1	32	119.031	90.573

187.045	start	0	32	119.031	90.573
188.577	leave	0	33	120.563	92.105
215.512	enter	1	33	120.563	92.105
215.512	start	0	33	120.563	92.105
216.445	leave	0	34	121.496	93.038
219.886	enter	1	34	121.496	93.038
219.886	start	0	34	121.496	93.038
220.045	leave	0	35	121.655	93.197
221.103	enter	1	35	121.655	93.197
221.103	start	0	35	121.655	93.197
221.341	leave	0	36	121.892	93.434
222.649	enter	1	36	121.892	93.434
222.649	start	0	36	121.892	93.434
224.895	leave	0	37	124.138	95.68
225.973	enter	1	37	124.138	95.68
225.973	start	0	37	124.138	95.68
226.514	leave	0	38	124.68	96.222
230.08	enter	1	38	124.68	96.222
230.08	start	0	38	124.68	96.222
230.158	enter	1	38	124.68	96.222

Carwash model run in Netsim:					
Uses TLNEX (TLN*1,000,000) values from SIGMA.					
Output vars time, WT, TE are then divided by 1,000,000 and rounded to 3 decimals.					
time	event	Q	CO	WT	TE
0	run	0	0	0	0
0	enter	1	0	0	0
0	start	0	0	0	0
0.844	leave	0	1	0.844	0.844
11.682	enter	1	1	0.844	0.844
11.682	start	0	1	0.844	0.844
15.884	leave	0	2	5.046	5.046
18.882	enter	1	2	5.046	5.046
18.882	start	0	2	5.046	5.046
20.381	enter	1	2	5.046	5.046
20.496	leave	1	3	6.660	6.660
20.496	start	0	3	6.660	6.660
23.759	leave	0	4	10.038	9.924
32.145	enter	1	4	10.038	9.924
32.145	start	0	4	10.038	9.924
34.154	enter	1	4	10.038	9.924
35.502	leave	1	5	13.395	13.280
35.502	start	0	5	13.395	13.280
36.057	leave	0	6	15.299	13.835
54.336	enter	1	6	15.299	13.835
54.336	start	0	6	15.299	13.835
64.418	leave	0	7	25.381	23.918
66.723	enter	1	7	25.381	23.918
66.723	start	0	7	25.381	23.918
67.187	leave	0	8	25.844	24.381
71.858	enter	1	8	25.844	24.381
71.858	start	0	8	25.844	24.381
79.149	leave	0	9	33.136	31.673
88.316	enter	1	9	33.136	31.673

88.316	start	0	9	33.136	31.673
88.572	enter	1	9	33.136	31.673
89.677	leave	1	10	34.497	33.034
89.677	start	0	10	34.497	33.034
89.771	enter	1	10	34.497	33.034
101.982	leave	1	11	47.907	45.339
101.982	start	0	11	47.907	45.339
102.839	leave	0	12	60.975	46.196
105.929	enter	1	12	60.975	46.196
105.929	start	0	12	60.975	46.196
106.722	enter	1	12	60.975	46.196
106.976	enter	2	12	60.975	46.196
107.343	leave	2	13	62.389	47.609
107.343	start	1	13	62.389	47.609
109.916	leave	1	14	65.583	50.183
109.916	start	0	14	65.583	50.183
114.851	enter	1	14	65.583	50.183
116.643	leave	1	15	75.250	56.910
116.643	start	0	15	75.250	56.910
117.844	leave	0	16	78.243	58.110
121.164	enter	1	16	78.243	58.110
121.164	start	0	16	78.243	58.110
122.160	enter	1	16	78.243	58.110
124.965	leave	1	17	82.044	61.911
124.965	start	0	17	82.044	61.911
128.122	leave	0	18	88.005	65.068
134.579	enter	1	18	88.005	65.068
134.579	start	0	18	88.005	65.068
139.684	leave	0	19	93.111	70.174
139.805	enter	1	19	93.111	70.174
139.805	start	0	19	93.111	70.174
141.037	leave	0	20	94.343	71.406
145.929	enter	1	20	94.343	71.406
145.929	start	0	20	94.343	71.406

148.252	leave	0	21	96.665	73.728
150.216	enter	1	21	96.665	73.728
150.216	start	0	21	96.665	73.728
152.603	leave	0	22	99.052	76.115
156.215	enter	1	22	99.052	76.115
156.215	start	0	22	99.052	76.115
156.825	leave	0	23	99.662	76.725
159.137	enter	1	23	99.662	76.725
159.137	start	0	23	99.662	76.725
159.382	enter	1	23	99.662	76.725
159.445	leave	1	24	99.970	77.033
159.445	start	0	24	99.970	77.033
161.218	leave	0	25	101.806	78.806
162.715	enter	1	25	101.806	78.806
162.715	start	0	25	101.806	78.806
162.892	leave	0	26	101.983	78.983
164.197	enter	1	26	101.983	78.983
164.197	start	0	26	101.983	78.983
164.248	enter	1	26	101.983	78.983
165.492	leave	1	27	103.278	80.279
165.492	start	0	27	103.278	80.279
166.471	enter	1	27	103.278	80.279
166.576	leave	1	28	105.606	81.362
166.576	start	0	28	105.606	81.362
167.062	leave	0	29	106.196	81.848
170.325	enter	1	29	106.196	81.848
170.325	start	0	29	106.196	81.848
173.420	enter	1	29	106.196	81.848
177.529	leave	1	30	113.401	89.052
177.529	start	0	30	113.401	89.052
177.958	leave	0	31	117.939	89.481
179.875	enter	1	31	117.939	89.481
179.875	start	0	31	117.939	89.481
180.967	leave	0	32	119.031	90.573

187.045	enter	1	32	119.031	90.573
187.045	start	0	32	119.031	90.573
188.578	leave	0	33	120.564	92.106
215.513	enter	1	33	120.564	92.106
215.513	start	0	33	120.564	92.106
216.446	leave	0	34	121.497	93.039
219.887	enter	1	34	121.497	93.039
219.887	start	0	34	121.497	93.039
220.045	leave	0	35	121.655	93.197
221.103	enter	1	35	121.655	93.197
221.103	start	0	35	121.655	93.197
221.341	leave	0	36	121.893	93.435
222.649	enter	1	36	121.893	93.435
222.649	start	0	36	121.893	93.435
224.895	leave	0	37	124.139	95.681
225.973	enter	1	37	124.139	95.681
225.973	start	0	37	124.139	95.681
226.515	leave	0	38	124.680	96.222
230.080	enter	1	38	124.680	96.222
230.080	start	0	38	124.680	96.222

Appendix F: Carwash Model: SIGMA Program

I. STATE VARIABLE DEFINITIONS.

For this simulation, the following state variables are defined:

S: (integer valued)
Q: (integer valued)
CI: (integer valued)
CO: (integer valued)
TS: (real valued)
TE: (real valued)
W[512]: (real valued)
WT: (real valued)

II. EVENT DEFINITIONS.

Simulation state changes are represented by event vertices (nodes or balls) in a SIGMA graph. Event vertex parameters, if any, are given in parentheses. Logical and dynamic relationships between pairs of events are represented in a SIGMA graph by edges (arrows) between event vertices. Unless otherwise stated, vertex execution priorities, to break time ties, are equal to 5.

1. The run() event causes the following state change(s):

S=1
Q=0
CI=0
CO=0
TS=0
TE=0
W=0
WT=0

After every occurrence of the run event:

Unconditionally, schedule the enter() event to occur without delay.

2. The enter() event causes the following state change(s):

Q=Q+1
CI=CI+1
W[CI]=CLK

After every occurrence of the enter event:

Unconditionally, schedule the enter() event to occur in -5*ln{rnd} time units.
(Time ties are broken by an execution priority of 4.)
If S>0, then schedule the start() event to occur without delay.

3. The start() event causes the following state change(s):
S=0
Q=Q-1
TS=CLK
After every occurrence of the start event:
Unconditionally, schedule the leave() event to occur in -3*ln{rnd} time units.
4. The leave() event causes the following state change(s):
S=1
CO=CO+1
TE=TE+CLK-TS
WT=WT+CLK-W[CO]
After every occurrence of the leave event:
If Q>0, then schedule the start() event to occur without delay.

Appendix G: Netsim Documentation

AnimationCanvas extends Canvas

Draws the event graph of the simulation model onto the viewing panel. Animates the event graph by periodically repainting sections as requested by the simulation thread, an instance of SchedThread.

AnimationCanvas must be contained in same text file as ViewerPanel.

Imports: java.awt.*, java.util.*.

Methods:

- **public AnimationCanvas(**DataDictionary data, ViewerPanel target, int width, int height**)**: Creates a new instance of AnimationCanvas associated with the instances identified by this method's parameters. Sets the size of this instance of AnimationCanvas.
- **private void backwardEdge(**int startIndex**)**: Draws an edge from the node indicated by the parameter "startIndex" to the preceding node.
  - Called by whichEdge(int, int, int).
- **private void curvedEdge(**int startIndex**)**: Draws an edge from the node indicated by the parameter "startIndex" back to the same node.
  - Called by whichEdge(int, int, int).
- **private void drawArrow(**Graphics g**)**: Draws an edge of the event graph in the direction and location specified by the associated instance variables.
  - Called by paint(Graphics).
  - Called by update(Graphics).
- **private void drawNode(**Graphics g**)**: Draws a node of the event graph in the location specified by the associated instance variables.
  - Called by paint(Graphics).
  - Called by update(Graphics).
- **private void forwardEdge(**int startIndex**)**: Draws an edge from the node indicated by the parameter" startIndex" to the next node.
  - Called by whichEdge(int, int, int).
- **private void locateNode(**int nodeIndex**)**: Locates the physical location of the current node.
  - Called by paint(Graphics).
  - Called by update(Graphics).
- **private void makeNodeList()**: Creates a list of the nodes for the event graph based on the events list in the database.
  - Called by paint(Graphics).
- **public void paint(**Graphics g**)**: Draws the initial event graph of the simulation model.
  - Overrides java.awt.Canvas.paint(Graphics).

- ◊ Calls drawArrow(Graphics).
- ◊ Calls drawNode(Graphics).
- ◊ Calls locateNode(int).
- ◊ Calls makeNodeList().
- ◊ Calls whichEdge(int, int, int).

♦ **public void repaint(**String[ ] event, Object[ ] item**)**: Locates the next section of the event graph to be animated.
- ◊ Called by SchedThread.run().
- ◊ Calls update(Graphics) by calling java.awt.Component.repaint(long).

♦ **public void update(**Graphics g**)**: Repaints the section of the model identified by repaint(String[ ], Object[ ]), causing the animation effect. From its original color of yellow, the section is painted cyan and then repainted yellow.
- ◊ Overrides java.awt.Component.update(Graphics).
- ◊ Calls drawArrow(Graphics).
- ◊ Calls drawNode(Graphics).
- ◊ Calls locateNode(int).
- ◊ Calls whichEdge(int, int, int).

♦ **private void whichEdge(**int next, int list, int active**)**: Uses the parameters of this method to determine the direction of the current edge and calls the appropriate method.
- ◊ Called by paint(Graphics).
- ◊ Called by update(Graphics).
- ◊ Calls backwardEdge(int).
- ◊ Calls curvedEdge(int).
- ◊ Calls forwardEdge(int).

CardPanel extends Panel

Allocates resources for each graphical user interface (GUI) and connects the interfaces for data transfer purposes.

Imports: java.awt.*, java.util.*.

Methods:

♦ **public CardPanel()**: Creates resources, much like hyper cards, for each user interface.
- ◊ Calls newCard(EntryPanel).
- ◊ Calls newCard(ViewerPanel).

♦ **private void fillEdgesHt(**MyHashtable edgesHt, String[ ] thisEventsKey, MyVector thisVedge**)**: Fills the edge hashtable with vectors, one for each edge.
- ◊ Called by transferDataS(EntryPanel, ViewerPanel).

- **private void fillEventsHt(**MyHashtable events, String[ ] thisEventsKey, MyHashtable thisVevents, String thisStateVars**)**: Fills the event hashtable with the event names and each individual event hashtable with the correct event name and state variables as listed on the entry form.
  - ◊ Called by transferDataS(EntryPanel, ViewerPanel).
  - ◊ Calls putVars(MyHashtable, String).
- **private Panel newCard(**EntryPanel thisModule**)**: Adds the entry form user interface, or card, to the main applet. Initializes and returns a started instance of this interface.
  - ◊ Called by CardPanel().
- **private Panel newCard(**ViewerPanel thisModule**)**: Adds the output viewer user interface, or card, to the main applet. Initializes and returns a started an instance of this interface.
  - ◊ Called by CardPanel().
- **private void putVars(**MyHashtable varHt, String varString**)**: Creates an instance of ModelParser to parse the equations for the state variables as entered in the entry form. Transfers the resulting information into the individual event hashtables.
  - ◊ Called by fillEventsHt(MyHashtable, String[ ], MyHashtable, String).
  - ◊ Calls ModelParser.decipherVars().
- **protected void transferDataS()**: Allows actual user interface instances to be hidden from the Main Applet.
  - ◊ Called by MainApplet.action(Event, Object).
  - ◊ Calls transferDataS(EntryPanel, ViewerPanel).
- **private void transferDataS(**EntryPanel epm, ViewerPanel vpm**)**: Reads the information from the entry form and transfers this data into the database.
  - ◊ Calls fillEdgesHt(MyHashtable, String[ ], MyVector).
  - ◊ Calls fillEventsHt(MyHashtable, String[ ], MyHashtable, String).
  - ◊ Calls placeEdge(MyHashtable, String, String, String, String, String).
- **private void placeEdge(**MyHashtable edgeHt, String fromEdge, String toEdge, String condEdge, String prEdge, String timeEdge**)**: Fills each edge vector with the information listed on the entry form for that edge.
  - ◊ Called by transferDataS(EntryPanel, ViewerPanel).

DataDictionary

Creates and manages the databases for the events, variables and future events list of the simulation model. Calls an instance of a random number stream as needed.
Events are maintained as keys in a hashtable with each element of the hashtable being a hashtable

holding the variables associated with that event key. Each element of the variable's hashtable is a string or integer representing the rule associated with that variable key. The edges of the event graph are also maintained in a hashtable with each element being a vector. Each element vector defines a property of that edge (i.e., location, condition, time delay, priority). The current values of the variables are maintained in a hashtable where the variable name is the key and the value is the element. The future events list is a vector with each element being a string array consisting of a time, priority, and event name.

Imports: java.util.*.

Methods:

- **public DataDictionary()**: Creates a new instance of DataDictionary.
- **protected Object[ ] adjustEventSch()**: Returns the next entry from the future events list to the simulation thread and removes that entry from the list.
  - Called by SchedThread.run().
- **private void clearData()**: Clears the databases of all previous data.
  - Called by putData(MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyVector, MyVector, MyVector, MyVector, MyVector, MyVector).
- **protected void initData(**MyHashtable initEvent**)**: Fills the initial variable values from the entry form into the variable database.
  - Called by ViewerPanel.initModel(String[ ], MyHashtable).
  - Called by setUp(long, MyHashtable).
- **private void makeDataTables(**MyHashtable incoming, MyHashtable toReplace**)**: Adds the event information to the databases.
  - Called by putData(MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyVector, MyVector, MyVector, MyVector, MyVector, MyVector).
- **private void makeDataTables(**MyVector incoming, MyVector toReplace**)**: Adds the variable information to the databases.
  - Called by putData(MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyVector, MyVector, MyVector, MyVector, MyVector, MyVector).
- **protected void putData(**MyHashtable htEV, MyHashtable ht1, MyHashtable ht2, MyHashtable ht3, MyHashtable ht4, MyHashtable ht5, MyHashtable ht6, MyHashtable htED, MyVector v1, MyVector v2, MyVector v3, MyVector v4, MyVector v5, MyVector v6**)**: Uses the following methods and the data from the entry form to fill in the databases.

- ◊ Called by ViewerPanel.putData(MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyVector, MyVector, MyVector, MyVector, MyVector, MyVector).
- ◊ Calls clearData().
- ◊ Calls makeDataTables(MyHashtable, MyHashtable).
- ◊ Calls makeDataTables(MyVector, MyVector).

♦ **protected void setUp(**long seedText, MyHashtable initEvent**)**: Calls the methods to clear the databases and future events list and to start a new random number stream. This method is called whenever a new simulation run is started.
- ◊ Called by SchedThread.SchedThread(DataDictionary, ViewerPanel, long, double).
- ◊ Calls initData(MyHashtable).
- ◊ Calls startRnd(long).

♦ **private void startRnd(**long theSeed**)**: Starts a new random number stream.
- ◊ Called by setUp(long, MyHashtable).

♦ **protected void updateEventSch(**String[ ] thisEvent, double currentTime**)**: Checks edge conditions between the current event and all connected events, schedules time for all valid connected events, and updates the future events list. Scheduling ties are broken by priorities, if given, with 9 being a higher priority and 1 being low. Otherwise, the newest event is placed below the existing, tying event.
- ◊ Called by SchedThread.run().
- ◊ Calls ModelParser.decipherCondition(MyHashtable).
- ◊ Calls RndGenerator.calcRndNum(String).

♦ **protected void updateEventVars(**MyHashtable varht, double currentTime**)**: Updates the values in the variables database according to the rules of the current event, given in the parameter "varht".
- ◊ Called by SchedThread.run().
- ◊ Calls ModelParser.decipherRule(Integer, Integer, MyHashtable, MyVector).

EntryPanel extends Panel implements Runnable

Defines and draws the graphical user interface (GUI) for defining and modifying the simulation model.

Imports: java.awt.*.

Methods:

♦ **public EntryPanel()**: Creates a new instance of EntryPanel.

- **private void labelEdgeSection(**String label**)**: Labels the text fields for defining the edges.
  - ◊ Called by run().
- **private void labelSection(**String label**)**: Labels the general sections of the entry form.
  - ◊ Called by run().
- **private void makeEdgeSection(**TextField from, TextField to, TextField cond, TextField td, TextField pr**)**: Places the text fields for the edge locations, conditions, time delays, and priorities.
  - ◊ Called by run().
- **private void makeEventSection(**String ename, TextField event, TextArea evar**)**: Labels and places the text fields for the event names and state variables.
  - ◊ Called by run().
- **public void run()**: Defines the layout and components of the entry form. Calls the methods that create and label these components.
  - ◊ Calls labelEdgeSection(String).
  - ◊ Calls labelSection(String).
  - ◊ Calls makeEdgeSection(TextField, TextField, TextField, TextField, TextField).
  - ◊ Calls makeEventSection(String, TextField, TextArea).
- **public void start()**: Starts a thread to manage the operations within this class. This allows Netsim to create GUI's simultaneously, reducing user waiting time for applet interpretation by the browser.
- **public void stop()**: Stops the thread managing this class.

MainApplet extends java.applet.Applet

Provides general applet behaviors for the Netsim program. This includes initializing the program; starting, stopping, and redrawing the applet as necessary; and destroying any resources used in the applet before closing.

Imports: java.awt.*.

Methods:

- **public boolean action(**Event evt, Object arg**)**: Handles actions required to switch between graphical user interfaces (GUI's) or to save the current GUI information (either model or output) into a file on the user's machine. The "save" buttons are currently non-functional.
  - ◊ Overrides java.awt.Component.action(Event, Object).
  - ◊ Called by user through mouse clicks on GUI button.
  - ◊ Calls CardPanel.transferDataS().

- **public void init()**: Sets the background formatting for the GUI's and initializes the default GUI. The current default is the entry form interface.
  - ◊ Overrides java.applet.Applet.init().
  - ◊ Calls insets().
  - ◊ Calls labelCard(String, String, String).
- **public Insets insets()**: Returns the amount of space between layout components and the border of the current container.
  - ◊ Overrides java.awt.Container.insets().
  - ◊ Called by init().
  - ◊ Calls java.awt.Insets(int, int, int, int).
- **private void labelCard(**String moduleLabel, String saveLabel, String buttonLabel**)**: Adjusts titles and button labels for a given graphical user interface (GUI).
  - ◊ Called by init().

ModelParser extends StringTokenizer

Parses textual information from the entry panel into forms compatible with the databases.
Imports: java.util.*.
Methods:

- **public ModelParser(**String rule**)**: Creates a new instance of the super class StringTokenizer with a blank space as the delimiter. This is used to parse edge conditions and variable rules.
- **public ModelParser(**String rule, String delim**)**: Creates a new instance of the super class StringTokenizer with the specified delimiter. This is used to parse random distributions and to separate and parse variable equations.
- **public void db(**String toDebug**)**: Prints statements to a standard output window for debugging purposes.
  - ◊ Called by various methods in ModelParser.
- **protected boolean decipherCondition(**MyHashtable ht**)**: Compares the edge condition to the current variable values and returns the booleam value of the condition, true or false.
  - ◊ Called by DataDictionary.updateEventSch(String[ ], double).
  - ◊ Calls tryEqual(Integer, int).
  - ◊ Calls tryGreater(Integer, int).
  - ◊ Calls tryLess(Integer, int).
  - ◊ Calls tryNot(Integer, int).
- **protected int[ ] decipherRndParam()**: Locates and returns the values of the parameters of the current random distribution.

- ◊ Called by RndGenerator.calcRndNum(String).

♦ **protected Integer decipherRule(**Integer old, Integer intTime, MyHashtable varData, MyVector W**)**: Parses the variable rule for the given variable and the current event. Calls the methods to change the current variable value, given by the parameter "old", by the appropriate amount. Returns the updated value to the database.
- ◊ Called by DataDictionary.updateEventVars(MyHashtable, double).
- ◊ Calls increaseVar(Integer, int).
- ◊ Calls replaceVar(int).

♦ **protected String[ ] decipherVars():** Deciphers variable equations, returning an array with the variable name and the variable value.
- ◊ Called by CardPanel.putVars(MyHashtable, String).

♦ **private Integer increaseVar(**Integer isNow, int howMuch**)**: Increases the current variable value by the specified amount, which may be positive or negative.
- ◊ Called by decipherRule(Integer, Integer, MyHashtable, MyVector).

♦ **private Integer replaceVar(**int howMuch**)**: Replaces the current variable value with the specified value.
- ◊ Called by decipherRule(Integer, Integer, MyHashtable, MyVector).

♦ **private boolean tryEqual(**Integer isNow, int compareTo**)**: Compares edge conditions consisting of an equality statement. Returns the boolean value of the statement.
- ◊ Called by decipherCondition(MyHashtable).

♦ **private boolean tryGreater(**Integer isNow, int compareTo**)**: Compares edge conditions consisting of a "greater than" statement. Returns the boolean value of the statement.
- ◊ Called by decipherCondition(MyHashtable).

♦ **private boolean tryLess(**Integer isNow, int compareTo**)**: Compares edge conditions consisting of a "less than" statement. Returns the boolean value of the statement.
- ◊ Called by decipherCondition(MyHashtable).

♦ **private boolean tryNot(**Integer isNow, int compareTo**)**: Compares edge conditions consisting of a "not equal" statement. Returns the boolean value of the statement.
- ◊ Called by decipherCondition(MyHashtable).

MyHashtable extends Hashtable

Extends java.util.Hashtable, purely for debugging purposes.
Imports: java.util.Hashtable, java.util.Enumeration.

Methods:

- **public MyHashtable(**int size**)**: Creates a new instance of the super class Hashtable with the specified size.
- **public void listAll()**: Lists the elements of a hashtable into a standard output window. This method is for hashtables where neither the keys nor elements are arrays.
  - Called by many methods in Netsim.
- **public void listAllE()**: Lists the elements of a hashtable into a standard output window. This method is for hashtables where the keys are arrays.
  - Called by many methods in Netsim.

MyVector extends Vector

Extends java.util.Vector, purely for debugging purposes.
Imports: java.util.Vector, java.util.Enumeration.
Methods:

- **public MyVector(**int size**)**: Creates a new instance of the super class Vector with the specified size.
- **public void listAll()**: Lists the elements of a vector into a standard output window. This method is for vectors where the elements are arrays of objects.
  - Called by many methods in Netsim.
- **public void listAllC()**: Lists the elements of a vector into a standard output window. This method is for vectors where the elements are arrays of characters.
  - Called by many methods in Netsim.

RndGenerator extends Random

Determines the random number stream for the current simulation run and calculates random variates from that stream as needed.
Imports: java.util.Random.
Methods:

- **public RndGenerator()**: Creates a new instance of the super class Random with a seed defined by the current system time of the computer.
- **public RndGenerator(**long theSeed**)**: Creates a new instance of the super class Random with the value of the parameter "theSeed" as the seed.
- **protected Double calcRndNum(**String description**)**: Takes the next number from the SIGMA random number stream, or SIGMA's associated set of natural logs, if the value of the seed is currently "-2". Otherwise, this method takes the next number of the random stream created by java.util.Random, using the current seed value. The method then notifies ModelParser to parse the parameter description

and calculates a random variate based on the distribution parameters of the resulting random distribution.

- ◊ Called by DataDictionary.updateEventSch(String[ ], double).
- ◊ Calls ModelParser.decipherRndParam().
- ◊ Calls sigmaRndNum().
- ◊ Calls sigmaNaturalLogOfRndNum().

- ♦ **private double sigmaRndNum()**: Supplies approx. the first 100 numbers in the random number stream created by SIGMA for the seed 12352.
  - ◊ Called by calcRndNum(String).
- ♦ **private double sigmaNaturalLogOfRndNum()**: Supplies the SIGMA calculated, natural logs for approx. the first 100 numbers in the random number stream created by SIGMA for the seed 12352.
  - ◊ Called by calcRndNum(String).

SchedThread extends Thread

Monitors the simulation clock and future event list, determines the type of output desired by the user, notifies the database of the current random number seed, and signals the database to process the next event. This class manages the simulation run until the clock exceeds the current run-length, the future events list is empty, or the simulation is stopped by the user.
SchedThread must be contained in same text file as ViewerPanel.
Imports: java.awt.*, java.util.*.
Methods:

- ♦ **public SchedThread(**DataDictionary data, ViewerPanel target, long seed, double runlength**)**: Creates a new instance of SchedThread associated with the instances identified by this method's parameters. Notifies the database to initialize the state variables. Identifies the starting node on the event graph.
  - ◊ Calls DataDictionary.setUp(long, MyHashtable).
- ♦ **public void run()**: Acts as the main method for the class, managing the simulation run and calling other methods as needed.
  - ◊ Calls DataDictionary.adjustEventSch().
  - ◊ Calls DataDictionary.updateEventSch(String[ ], double).
  - ◊ Calls DataDictionary.updateEventVars(MyHashtable, double).
  - ◊ Calls outputData().
  - ◊ Calls AnimationCanvas.repaint(String[ ], Object[ ]).
  - ◊ Calls ViewerPanel.stopN().
- ♦ **public void outputData()**: Writes the data from the simulation run into the text area of the viewing interface or into a standard output text window associated with the user's browser.

- ◊ Called by run().

ViewerPanel extends Panel implements Runnable

Defines and draws the GUI for viewing and interacting with the simulation model.
Imports: java.awt.*, java.util.*.
Methods:

- ♦ **public ViewerPanel()**: Creates a new instance of ViewerPanel.
- ♦ **public boolean action(**Event evt, Object arg**)**: Handles button actions for interacting with this GUI: stop, pause or play.
  - ◊ Overrides java.awt.Component.action(Event, Object).
  - ◊ Called by user through mouse clicks on GUI button.
  - ◊ Calls stopB().
  - ◊ Calls pauseB().
  - ◊ Calls playB().
- ♦ **protected void initModel(**String[ ] startNode, MyHashtable initEvent**)**: Notifies the database of the first event in the model.
  - ◊ Calls DataDictionary.initData(MyHashtable).
- ♦ **public Insets insets()**: Sets the amount of space between layout components and the border of the current container.
  - ◊ Overrides java.awt.Container.insets().
  - ◊ Called by run().
  - ◊ Calls java.awt.Insets(int, int, int, int).
- ♦ **protected void putData(**MyHashtable htEV, MyHashtable ht1, MyHashtable ht2, MyHashtable ht3, MyHashtable ht4, MyHashtable ht5, MyHashtable ht6, MyHashtable htED, MyVector v1, MyVector v2, MyVector v3, MyVector v4, MyVector v5, MyVector v6**)**: Passes the data from the entry form to the database.
  - ◊ Calls DataDictionary.putData(MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyHashtable, MyVector, MyVector, MyVector, MyVector, MyVector, MyVector).
- ♦ **private void pauseB()**: Pauses simulation thread when user clicks the “pause” button.
  - ◊ Called by action().
- ♦ **private void playB()**: Begins or resumes simulation thread when user clicks the “play” button.
  - ◊ Called by action().
  - ◊ Calls setSD().

- ◊ Calls setRL().

♦ **private double readRunLength(**String lengthChangeTo**)**: Reads the value in the "model run-length" text field or calculates a new value, if necessary, and returns this value.
- ◊ Called by setRL().

♦ **private long readSeed(**String seedChangeTo**)**: Reads the value in the "random number seed" text field or calculates a new value, if necessary, and returns this value.
- ◊ Called by setSD().

♦ **public void run()**: Defines the layout and components of the viewing panel and also creates and labels these components.
- ◊ Calls insets().

♦ **public void start()**: Starts a thread to manage the operations within this class. This allows Netsim to create GUI's simultaneously, reducing user waiting time for applet interpretation by the browser.

♦ **private double setRL()**: Replaces the value in the "model run-length" text field with the current value and returns the current value.
- ◊ Called by playB().
- ◊ Calls readRunLength(String).

♦ **private long setSD()**: Replaces the value in the "random number seed" text field with the current value and returns the current value.
- ◊ Called by playB().
- ◊ Calls readSeed(String).

♦ **public void stop()**: Stops the thread managing this class.

♦ **private void stopB()**: Stops simulation thread when user clicks the "stop" button.
- ◊ Called by action().

♦ **protected void stopN()**: Stops simulation thread when simulation ends naturally. Called by SchedThread.run().

Appendix H: Netsim Source Code

MainApplet HTML Document

```
<HTML>
<HEAD>
<TITLE>TISA</TITLE>
</HEAD>

<BODY>
Tamie's Internet Simulation Applet for an M/M/1 queue.<br>
<img src="../eyes.gif"><br>
Please be patient while your computer downloads and interprets the applet.<br>
<APPLET CODE = "netSim/MainApplet.class" WIDTH =600 HEIGHT =510>
<center>
This space is where the applet should be.<br>
Please check your operating system and internet browser for Java viewing
capabilities.<br>
Learn more about Java at <a href="http://www.javasoft.com">www.javasoft.com</a>
</center>
</APPLET>
</BODY>

</HTML>
```

```
1   package netSim;
2   import java.awt.*;
3   import java.util.*;
4
5   /**
6           CardPanel extends Panel
7           Allocates resources for each graphical user interface (GUI) and connects
8           the interfaces for data transfer purposes.
9   */
10
11  class CardPanel extends Panel{
12          EntryPanel entry;
13          ViewerPanel view;
14
15          public CardPanel() {
16                  entry = new EntryPanel();
17                  view = new ViewerPanel();
18                  setLayout(new CardLayout());
19                  add("Entry Form",newCard(entry));
20                  add("Simulator", newCard(view));
21          }
22
23          private Panel newCard(EntryPanel thisModule) {
24                  Panel parent = new Panel();
25                  parent.add("Center", thisModule);
26                  thisModule.start();
27                  return parent;
28          }
29
30          private Panel newCard(ViewerPanel thisModule) {
31                  Panel parent = new Panel();
32                  parent.add("Center", thisModule);
33                  thisModule.start();
34                  return parent;
35          }
36
37          protected void transferDataS() {  //called by MainApplet.action(,)
38                  transferDataS(entry,view);
39          }
40
```

```
41          private void transferDataS(EntryPanel epm, ViewerPanel vpm) {
42                  /*init each event*/
43                          String[] eventsKey1 = {new String("0"), epm.event1.getText()};
44                          String[] eventsKey2 = {new String("1"), epm.event2.getText()};
45                          String[] eventsKey3 = {new String("2"), epm.event3.getText()};
46                          String[] eventsKey4 = {new String("3"), epm.event4.getText()};
47                          String[] eventsKey5 = {new String("4"), epm.event5.getText()};
48                          String[] eventsKey6 = {new String("5"), epm.event6.getText()};
49                  /*create main &individual event hts*/
50                          MyHashtable events = new MyHashtable(10);
51                          MyHashtable vevent1 = new MyHashtable(10);
52                          MyHashtable vevent2 = new MyHashtable(10);
53                          MyHashtable vevent3 = new MyHashtable(10);
54                          MyHashtable vevent4 = new MyHashtable(10);
55                          MyHashtable vevent5 = new MyHashtable(10);
56                          MyHashtable vevent6 = new MyHashtable(10);
57                  /*fill events ht & individual event hts*/
58                          fillEventsHt(events, eventsKey1, vevent1, epm.evars1.getText());
59                          fillEventsHt(events, eventsKey2, vevent2, epm.evars2.getText());
60                          fillEventsHt(events, eventsKey3, vevent3, epm.evars3.getText());
61                          fillEventsHt(events, eventsKey4, vevent4, epm.evars4.getText());
62                          fillEventsHt(events, eventsKey5, vevent5, epm.evars5.getText());
63                          fillEventsHt(events, eventsKey6, vevent6, epm.evars6.getText());
64                  /*create main & individual  edge vects*/
65                          MyHashtable edges = new MyHashtable(10);
66                          MyVector vedge1 = new MyVector(8);
67                          MyVector vedge2 = new MyVector(8);
68                          MyVector vedge3 = new MyVector(8);
69                          MyVector vedge4 = new MyVector(8);
70                          MyVector vedge5 = new MyVector(8);
71                          MyVector vedge6 = new MyVector(8);
72                  /*fill edges ht*/
73                          fillEdgesHt(edges, eventsKey1, vedge1);
74                          fillEdgesHt(edges, eventsKey2, vedge2);
75                          fillEdgesHt(edges, eventsKey3, vedge3);
76                          fillEdgesHt(edges, eventsKey4, vedge4);
77                          fillEdgesHt(edges, eventsKey5, vedge5);
78                          fillEdgesHt(edges, eventsKey6, vedge6);
79                  /*fill individual edge vectors*/
80                          placeEdge(edges, epm.from1.getText(), epm.to1.getText(),
```

```
                                epm.cond1.getText(), epm.pr1.getText(),
                                epm.td1.getText());
                        placeEdge(edges, epm.from2.getText(), epm.to2.getText(),
                                epm.cond2.getText(), epm.pr2.getText(),
                                 epm.td2.getText());
                        placeEdge(edges, epm.from3.getText(), epm.to3.getText(),
                                epm.cond3.getText(), epm.pr3.getText(),
                                epm.td3.getText());
                        placeEdge(edges, epm.from4.getText(), epm.to4.getText(),
                                epm.cond4.getText(), epm.pr4.getText(),
                                 epm.td4.getText());
                        placeEdge(edges, epm.from5.getText(), epm.to5.getText(),
                                epm.cond5.getText(), epm.pr5.getText(),
                                epm.td5.getText());
                        placeEdge(edges, epm.from6.getText(), epm.to6.getText(),
                                epm.cond6.getText(), epm.pr6.getText(),
                                 epm.td6.getText());
                //events.listAllE(); //debug
                //edges.listAllE();  //debug
                //put data into ViewerPanel/DataDictionary;
                        vpm.putData(events, vevent1, vevent2, vevent3, vevent4, vevent5,
                                vevent6, edges, vedge1, vedge2, vedge3, vedge4, vedge5,
                                 vedge6);
                        vpm.initModel(eventsKey1, vevent1);
        }

        private void fillEventsHt(MyHashtable events, String[] thisEventsKey,
                        MyHashtable thisVevent, String thisStateVars) {
                if (!thisEventsKey[1].equals("")) {
                        events.put(thisEventsKey, thisVevent);
                        putVars(thisVevent, thisStateVars);
                }
        }

        private void fillEdgesHt(MyHashtable edgesHt, String[] thisEventsKey,
                        MyVector thisVedge) {
                if (!thisEventsKey[1].equals("")) {
                        edgesHt.put(thisEventsKey, thisVedge);
                }
        }
```

```
121
122             private void putVars(MyHashtable varHt, String varString) {
123                     ModelParser mp= new ModelParser(varString, ";");
124                     String[] stateVars = mp.decipherVars();
125                     //System.out.println(stateVars);
126                     for (int i=0; i < stateVars.length; i+=2) {
127                             String thisKey = stateVars[i];
128                             String thisElem = stateVars[i+1];
129                             varHt.put(thisKey,thisElem);
130                     }
131             }
132
133             private void placeEdge(MyHashtable edgeHT, String fromEdge, String toEdge,
134                                             String condEdge, String prEdge, String timeEdge) {
135                     for (Enumeration ef = edgeHT.elements(), kf = edgeHT.keys();
136                                     ef.hasMoreElements();){
137                             Vector elemf = (Vector) ef.nextElement();
138                             String[] keyf = (String[]) kf.nextElement();
139                             if (keyf[1].equals(fromEdge)) {
140                                     for (Enumeration et = edgeHT.elements(),
141                                              kt = edgeHT.keys();  et.hasMoreElements();){
142                                             Vector elemt = (Vector) et.nextElement();
143                                             String[] keyt = (String[]) kt.nextElement();
144                                             if (keyt[1].equals(toEdge)) {
145                                                     elemf.addElement(keyt);//set edge conditions
146                                                     elemf.addElement(condEdge);
147                                                     try {
148                                                     elemf.addElement(new Integer(prEdge));
149                                                     } catch(NumberFormatException nfe) {
150                                                             elemf.addElement(new Integer(5));
151                                                     }
152                                                     elemf.addElement(timeEdge);
153                                                     break;
154                                             }
155                                     }
156                                     break;
157                             }
158                     }
159             }
160     } //endof CardPanel class
```

```
1   package netSim;
2   import java.util.*;
3   import java.lang.*;
4
5   /**
6           DataDictionary
7           Creates and manages the databases for the events, variables and future
8           events list of the simulation model.  Calls an instance of a random number
9           stream as needed.
10          Events are maintained as keys in a hashtable with each element of the
11          hashtable being a hashtable holding the variables associated with that
12          event key.  Each element of the variable's hashtable is a string or integer
13          representing the rule associated with that variable key.  The edges of the
14          event graph are also maintained in a hashtable with each element being a
15          vector.  Each element vector defines a property of that edge (i.e.,
16          location, condition, time delay, priority).  The current values of the
17          variables are maintained in a hashtable where the variable name is the key
18          and the value is the element.  The future events list is a vector with each
19          element being a string array consisting of a time, priority, and event name.
20  */
21
22  class DataDictionary{
23          MyHashtable events = new MyHashtable(10);       //events
24          MyHashtable vevent1 = new MyHashtable(10);      //vars,each assoc w an event
25          MyHashtable vevent2 = new MyHashtable(10);
26          MyHashtable vevent3 = new MyHashtable(10);
27          MyHashtable vevent4 = new MyHashtable(10);
28          MyHashtable vevent5 = new MyHashtable(10);
29          MyHashtable vevent6 = new MyHashtable(10);
30          MyHashtable edges = new MyHashtable(10);                //edges
31          MyVector vedge1 = new MyVector(8);      //conditions,each assoc with an edge
32          MyVector vedge2 = new MyVector(8);
33          MyVector vedge3 = new MyVector(8);
34          MyVector vedge4 = new MyVector(8);
35          MyVector vedge5 = new MyVector(8);
36          MyVector vedge6 = new MyVector(8);
37          MyHashtable varData = new MyHashtable(10);      //main variable database
38          MyVector linkedList = new MyVector(10);
39                          //scheduling event list, elements are String[3]
40          RndGenerator RandomStream;                      //Random Stream var
```

```
41              double time;
42              MyVector W = new MyVector(500); //vector for customerIn counting
43
44              public DataDictionary() {
45              }
46
47              protected void putData(MyHashtable htEV, MyHashtable ht1, MyHashtable
48      ht2,
49                      MyHashtable ht3, MyHashtable ht4, MyHashtable ht5, MyHashtable
50      ht6,
51                              MyHashtable htED, MyVector v1, MyVector v2, MyVector v3,
52                              MyVector v4, MyVector v5, MyVector v6){
53                      clearData();
54                      makeDataTables(htEV, events);
55                      makeDataTables(ht1, vevent1);
56                      makeDataTables(ht2, vevent2);
57                      makeDataTables(ht3, vevent3);
58                      makeDataTables(ht4, vevent4);
59                      makeDataTables(ht5, vevent5);
60                      makeDataTables(ht6, vevent6);
61
62                      makeDataTables(htED, edges);
63                      makeDataTables(v1, vedge1);
64                      makeDataTables(v2, vedge2);
65                      makeDataTables(v3, vedge3);
66                      makeDataTables(v4, vedge4);
67                      makeDataTables(v5, vedge5);
68                      makeDataTables(v6, vedge6);
69              }
70
71              private void clearData() {
72                      vevent1.clear();
73                      vevent2.clear();
74                      vevent3.clear();
75                      vevent4.clear();
76                      vevent5.clear();
77                      vevent6.clear();
78                      events.clear();
79                      vedge1.removeAllElements();
80                      vedge2.removeAllElements();
```

```
81                  vedge3.removeAllElements();
82                  vedge4.removeAllElements();
83                  vedge5.removeAllElements();
84                  vedge6.removeAllElements();
85                  edges.clear();
86          }
87
88          private void makeDataTables(MyHashtable incoming, MyHashtable toReplace) {
89                  for (Enumeration e = incoming.elements(), k = incoming.keys();
90                                  e.hasMoreElements();){
91                          toReplace.put(k.nextElement(), e.nextElement());
92                  }
93          }
94
95          private void makeDataTables(MyVector incoming, MyVector toReplace) {
96                  for (Enumeration e = incoming.elements();e.hasMoreElements();){
97                          toReplace.addElement(e.nextElement());
98                  }
99          }
100
101         protected void initData(MyHashtable initEvent) {
102                         //set init var values, called by ViewerPanel
103                 for (Enumeration e = initEvent.elements(), k = initEvent.keys();
104                                 e.hasMoreElements();){
105                         String elem = (String) e.nextElement();
106                         String key = (String) k.nextElement();
107                         try {
108                                 varData.put(key, new Integer(elem));
109                         } catch (NumberFormatException exp) {}
110                         for (int i=0; i<501; i++) {
111                                 W.addElement(new Integer(0));
112                         }
113                 }
114         }
115
116         protected void setUp(long seedText, MyHashtable initEvent) {
117                         //used by SchedThread (ViewerPanel)
118                 varData.clear();  //empty var database
119                 initData(initEvent);    //set init var values
120                 linkedList.removeAllElements(); //empty linkedList
```

```
121                 startRnd(seedText); //restart Rnd Num Stream
122         }
123
124         private void startRnd(long theSeed){
125                 RandomStream = new RndGenerator(theSeed);
126         }
127
128         protected void updateEventVars(MyHashtable varht, double currentTime) {
129                 String stringIndex = null;
130                 int intIndex = 0;
131                 Double dblTime= new Double(currentTime*10000);
132                 Integer intTime = new Integer(dblTime.intValue());
133                 for (Enumeration e1 = varht.elements(), k1 = varht.keys();
134                                 e1.hasMoreElements();){
135                         String elem1 = (String) e1.nextElement();
136                         String key1 = (String) k1.nextElement();
137                         //update all non array vars, before array vars!!!
138                                 for (Enumeration e2 = varData.elements(),
139                                         k2 = varData.keys();   e2.hasMoreElements();){
140                                         Integer elem2= (Integer) e2.nextElement();
141                                         String key2 = (String) k2.nextElement();
142                                         //System.out.print("key1: "+key1+" elem1:
143                                         //                      "+elem1); //debug
144                                         //System.out.println(" key2 "+key2+ " elem2:
145                                         //                      "+elem2);//debug
146                                         if (key2.equals(key1)) {
147                                           ModelParser rg= new ModelParser(elem1);
148                                           Integer theUpdateValue = rg.decipherRule(elem2,
149                                                                   intTime, varData, W);
150                                           //System.out.println("update "+key2+" ToThis "
151                                            //      + theUpdateValue); //debug
152                                            varData.put(key2,theUpdateValue);
153                                           break;
154                                         }
155                                 }
156                 }
157                 for (Enumeration eA = varht.elements(), kA = varht.keys();
158                                 eA.hasMoreElements();){
159                         String elemA = (String) eA.nextElement();
160                         String keyA = (String) kA.nextElement();
```

```
161                             // now update array vars
162                                     if (keyA.startsWith("W[")) {
163                                             //parse out into vector
164                                             stringIndex = keyA.substring(2,(keyA.length() -1));
165                                             for (Enumeration e3 = varData.elements(),
166                                                     k3 = varData.keys(); e3.hasMoreElements();){
167                                                Integer elem3= (Integer) e3.nextElement();
168                                                String key3 = (String) k3.nextElement();
169                                                if (key3.equals(stringIndex)) {
170                                                        intIndex = elem3.intValue();
171                                                        break;
172                                                }
173                                             }
174                                             if (W.size() > intIndex) {
175                                                W.setElementAt(intTime,intIndex);          //force
176     W = clk
177                                             } else {
178                                                     W.addElement(intTime);        //force W = clk
179                                             }
180                                     }
181                     }
182             } //endof updateEventVars
183
184             protected void updateEventSch(String[] thisEvent, double currentTime) {
185                     double time = currentTime;
186                     Double schEventIn, schEventAt;
187                     String[] nextEvent;
188                     Integer priority;
189                     String whenEvent;
190                     Object[] tmp;
191                     Double compareEventTo;
192                     Integer comparePriorityTo;
193                     int index = 0;
194                     for (Enumeration e = edges.elements(), k = edges.keys();
195                                     e.hasMoreElements();){
196                             Vector elem = (Vector) e.nextElement();
197                             String[] key = (String[]) k.nextElement();
198                             //System.out.print("key "+key[0]+", "+key[1]+", thisEvent  ");
199                             //System.out.println(thisEvent[0]+", "+thisEvent[1]);
200                             if (key.equals(thisEvent)) {
```

```
                                    for (int i = 0;i < elem.size();i = i+4) {
                                      String checkCondition = (String) elem.elementAt(i+1);
                                      //System.out.println("checkCondition " +
checkCondition+
                                      //" boolean value is:
                                      //"+Boolean.getBoolean(checkCondition));  //debug
                                      ModelParser rg= new ModelParser(checkCondition);
                                      boolean theResult = rg.decipherCondition(varData);
                                      //System.out.println("theResult "+theResult); //debug
                        if (theResult) {
                                nextEvent = (String[]) elem.elementAt(i);
                                priority = (Integer) elem.elementAt(i+2);
                                whenEvent = (String) elem.elementAt(i+3);
                                schEventIn = RandomStream.calcRndNum(whenEvent);
                                schEventAt= new  Double(schEventIn.doubleValue()+time);
                                //System.out.println("nextEvent:" + nextEvent[0]+", "
                                // +nextEvent[1]); //debug
                                //System.out.println("priority " + priority);//debug
                                //System.out.println("schEventAt " + schEventAt); //debug
                                Object[] LLelement = new Object[3];
                                LLelement[0] = schEventAt;
                                LLelement[1] = priority;
                                LLelement[2] = nextEvent;
                                //System.out.println("LLelement:" + " schEventAt is "
                                //+LLelement[0]+ ", priority is "+LLelement[1]+
                                //", nextEvent is "+nextEvent[0]+","+nextEvent[1]);//debug
                                if (linkedList.isEmpty()) {
                                        linkedList.addElement(LLelement);
                                        //System.out.println("(first) linkedList: ");
                                        //linkedList.listAll(); //debug
                                }
                                else {
                                  index = 0;
                                  for (Enumeration eLL =linkedList.elements();
                                        eLL.hasMoreElements();) {
                                        tmp = (Object[]) eLL.nextElement();
                                        compareEventTo = (Double) tmp[0];
                                       comparePriorityTo = (Integer) tmp[1];
                                        if ((schEventAt.doubleValue() <
                                                compareEventTo.doubleValue())
```

```
241                                         |
242                                         ((schEventAt.doubleValue() ==
243                                         compareEventTo.doubleValue()) &
244                                          (priority.intValue()       >
245                                         comparePriorityTo.intValue()))
246                                         ) {
247                                          linkedList.insertElementAt(LLelement,index);
248                                          break;
249                                 }
250                                 else if (index == linkedList.size() -1) {
251                                         linkedList.addElement(LLelement);
252                                         break;
253                                 }
254                                 index = index+1;
255                         }
256                         //System.out.println("linkedList: "); //debug
257                         //linkedList.listAll(); //debug
258                                                 }
259                                         }
260                                 }
261                                 break;
262                         }
263                 }
264         } //endof updateEventSch
265
266         protected Object[] adjustEventSch() {
267                 Object [] LLmarker=null;
268                 try {
269                         LLmarker = (Object[]) linkedList.firstElement();
270                         linkedList.removeElementAt(0);
271                 } catch (NoSuchElementException e) {
272                         System.out.println("linkedList was empty");}
273                 return(LLmarker);
274         }
275 } //endof DataDictionary class
```

```
1   package netSim;
2   import java.awt.*;
3
4   /**
5           EntryPanel extends Panel implements Runnable
6           Defines and draws the graphical user interface (GUI) for defining and
7           modifying the simulation model.
8   */
9
10  class EntryPanel extends Panel implements Runnable{
11          GridBagLayout gridbag;
12          GridBagConstraints gbc;
13          Thread entryThread;
14
15          /*define events*/
16                  TextField event1,event2,event3,event4,event5,event6;
17          /*define state vars for each event*/
18                  TextArea evars1,evars2,evars3,evars4,evars5,evars6;
19          /*define edges*/
20                  TextField from1,from2,from3,from4,from5,from6;
21                  TextField to1,to2,to3,to4,to5,to6;
22                  TextField pr1,pr2,pr3,pr4,pr5,pr6;
23                  TextField cond1,cond2,cond3,cond4,cond5,cond6;
24                  TextField td1,td2,td3,td4,td5,td6;
25
26          public EntryPanel() {
27          }
28
29          public void start() {
30                  if (entryThread == null) {
31                          entryThread = new Thread(this);
32                          entryThread.start();
33                  }
34          }
35
36          public void run() {
37                  /*define events*/
38                          event1 = new TextField("run",7);
39                          event2 = new TextField("enter",7);
40                          event3 = new TextField("start",7);
```

```
41                      event4 = new TextField("leave",7);
42                      event5 = new TextField(7);
43                      event6 = new TextField(7);
44              /*define state vars for each event*/
45              evars1 = new
46      TextArea("S=1;Q=0;CI=0;CO=0;TS=0;TE=0;W[]=0;WT=0", 2,10);
47              evars2 = new TextArea("Q=+ 1;CI=+ 1;W[CI]=clk", 2,10);
48              evars3 = new TextArea("S=0;Q=- 1;TS=clk", 2,10);
49              evars4 = new TextArea("S=1;CO=+ 1;TE=+ clk - TS;WT=+ clk -\
50      W[CO]",2,10);
51              evars5 = new TextArea(2,10);
52              evars6 = new TextArea(2,10);
53              /*define edges*/
54                      from1 = new TextField("run", 7);
55                      to1 = new TextField("enter", 7);
56                      pr1 = new TextField("5",1);
57                      cond1 = new TextField("TRUE",10);
58                      td1 = new TextField("", 10);
59                      from2 = new TextField("enter", 7);
60                      to2 = new TextField("enter", 7);
61                      pr2 = new TextField("6",1);
62                      cond2 = new TextField("TRUE", 10);
63                      td2 = new TextField("exp(5)", 10);
64                      from3 = new TextField("enter", 7);
65                      to3 = new TextField("start", 7);
66                      pr3 = new TextField("5",1);
67                      cond3 = new TextField("S > 0",10);
68                      td3 = new TextField("",10);
69                      from4 = new TextField("start", 7);
70                      to4 = new TextField("leave", 7);
71                      pr4 = new TextField("5",1);
72                      cond4 = new TextField("TRUE",10);
73                      td4 = new TextField("exp(3)", 10);
74                      from5 = new TextField("leave", 7);
75                      to5 = new TextField("start", 7);
76                      pr5 = new TextField("5",1);
77                      cond5 = new TextField("Q > 0",10);
78                      td5 = new TextField("", 10);
79                      from6 = new TextField(7);
80                      to6 = new TextField(7);
```

```
81                      pr6 = new TextField(1);
82                      cond6 = new TextField(10);
83                      td6 = new TextField(10);
84              /*set fonts*/
85                      Font f = new Font("TimesRoman",Font.PLAIN,10);
86                      Font bf = new Font("TimesRoman",Font.BOLD,12);
87              /*setlayout*/
88                      gridbag = new GridBagLayout();
89                      gbc = new GridBagConstraints();
90                      setLayout(gridbag);
91                      gbc.ipadx = 2;
92                      gbc.ipady=1;
93                      setFont(bf);
94                      gbc.gridx = 1;
95                      gbc.gridy = 1;
96                      gbc.gridwidth = 2;
97                      gbc.anchor= GridBagConstraints.NORTHWEST;
98                      labelSection("EVENTS");
99                      gbc.gridx = 1;
100                     gbc.gridy = 2;
101                     gbc.gridwidth = 2;
102                     gbc.anchor= GridBagConstraints.NORTHWEST;
103                     labelSection("& VARS");
104                     gbc.gridx = 1;
105                     gbc.gridy = 3;
106                     gbc.gridwidth = 2;
107                     labelSection("(nodes &");
108                     gbc.gridx = 1;
109                     gbc.gridy = 4;
110                     gbc.gridwidth = 2;
111                     labelSection("attributes):");
112             /*event textboxes*/
113                     setFont(f);
114                     gbc.gridx = 3;
115                     gbc.gridy = 1;
116                     gbc.gridwidth = 1;
117                     makeEventSection("event 1",event1,evars1);
118                     makeEventSection("event 2",event2,evars2);
119                     makeEventSection("event 3",event3,evars3);
120                     gbc.gridx = 3;
```

```
121                         gbc.gridy = 3;
122                         makeEventSection("event 4",event4,evars4);
123                         makeEventSection("event 5",event5,evars5);
124                         makeEventSection("event 6",event6,evars6);
125                 /*create new panel*/
126                         setFont(bf);
127                         gbc.gridx = 1;
128                         gbc.gridy = 5;
129                         gbc.gridwidth = 2;
130                         gbc.insets = new Insets(10,0,0,0);
131                         gbc.anchor= GridBagConstraints.NORTHWEST;
132                         labelSection("EDGES:");
133                         setFont(f);
134                         gbc.gridx = 3;
135                         gbc.gridy = 5;
136                         gbc.gridwidth = 1;
137                         gbc.insets = new Insets(1,0,5,0);
138                         labelEdgeSection("from");
139                         labelEdgeSection("to");
140                         labelEdgeSection("condition");
141                         labelEdgeSection("time delay");
142                         labelEdgeSection("priority");
143                         gbc.gridx = 3;
144                         gbc.gridy = 6;
145                         makeEdgeSection(from1,to1,cond1,td1,pr1);
146                         makeEdgeSection(from2,to2,cond2,td2,pr2);
147                         makeEdgeSection(from3,to3,cond3,td3,pr3);
148                         makeEdgeSection(from4,to4,cond4,td4,pr4);
149                         makeEdgeSection(from5,to5,cond5,td5,pr5);
150                         makeEdgeSection(from6,to6,cond6,td6,pr6);
151                         gbc.gridx = 1;
152                         gbc.gridy = 12;
153         }
154
155         public void stop() {
156                 if (entryThread != null) {
157                         entryThread.stop();
158                         entryThread = null;
159                 }
160         }
```

```
161
162             private void labelSection(String label) {
163                     Label L1 = new Label();
164                     L1.setForeground(Color.blue);
165                     L1.setText(label);
166                     gridbag.setConstraints(L1,gbc);
167                     add(L1);
168                     gbc.gridx = gbc.gridx+1;
169             }
170
171             private void makeEventSection(String ename,TextField event,TextArea evar) {
172                     gbc.insets = new Insets(1,0,5,0);
173                     gbc.anchor= GridBagConstraints.SOUTH;
174                     Label L1 = new Label();
175                     L1.setText(ename);
176                     gridbag.setConstraints(L1,gbc);
177                     add(L1);
178                     gbc.insets = new Insets(1,0,3,0);
179                     gbc.gridx = gbc.gridx+1;
180                     Label L2 = new Label("state vars");
181                     gridbag.setConstraints(L2,gbc);
182                     add(L2);
183                     gbc.insets = new Insets(1,0,5,0);
184                     gbc.anchor= GridBagConstraints.NORTH;
185                     gbc.gridx = gbc.gridx-1;
186                     gbc.gridy = gbc.gridy+1;
187                     gridbag.setConstraints(event,gbc);
188                     add(event);
189                     gbc.insets = new Insets(1,0,3,0);
190                     gbc.gridx = gbc.gridx+1;
191                     gridbag.setConstraints(evar,gbc);
192                     add(evar);
193                     gbc.gridx = gbc.gridx+1;
194                     gbc.gridy = gbc.gridy-1;
195             }
196
197             private void labelEdgeSection(String label) {
198                     gbc.anchor= GridBagConstraints.SOUTH;
199                     Label L1 = new Label();
200                     L1.setText(label);
```

```
201                     gridbag.setConstraints(L1,gbc);
202                     add(L1);
203                     gbc.gridx = gbc.gridx+1;
204             }
205
206             private void makeEdgeSection(TextField from,TextField to,TextField cond,
207                                                     TextField td,TextField pr){
208                     gbc.insets = new Insets(0,1,5,0);
209                     gbc.anchor= GridBagConstraints.NORTH;
210                     gridbag.setConstraints(from,gbc);
211                     add(from);
212                     gbc.gridx = GridBagConstraints.RELATIVE;
213                     gbc.insets = new Insets(0,1,3,0);
214                     gridbag.setConstraints(to,gbc);
215                     add(to);
216                     gbc.gridx = GridBagConstraints.RELATIVE;
217                     gbc.insets = new Insets(0,1,5,0);
218                     gridbag.setConstraints(cond,gbc);
219                     add(cond);
220                     gbc.gridx = GridBagConstraints.RELATIVE;
221                     gbc.insets = new Insets(0,1,5,0);
222                     gridbag.setConstraints(td,gbc);
223                     add(td);
224                     gbc.gridx = GridBagConstraints.RELATIVE;
225                     gbc.insets = new Insets(0,1,5,0);
226                     gridbag.setConstraints(pr,gbc);
227                     add(pr);
228                     gbc.gridx = 3;
229                     gbc.gridy = gbc.gridy+1;
230             }
231     } //endof EntryPanel class
```

```
1   package netSim;
2   import java.awt.*;
3
4   /**
5           MainApplet extends java.applet.Applet
6           Provides general applet behaviors for the Netsim program. This
7           includes initializing the program; starting, stopping, and redrawing
8           the applet as necessary; and destroying any resources used in the
9           applet before closing.
10  */
11
12  public class MainApplet extends java.applet.Applet {
13          Font f = new Font("TimesRoman",Font.PLAIN,12);
14          Panel pn = new Panel();                         //north panel
15          Panel pc = new Panel();                         //center panel
16          Panel ps = new Panel();                         //south panel
17          Label title = new Label();                  //applet title
18          Button saveCard = new Button();             //save button
19          Button switchCard = new Button();   //switch between views
20          CardPanel viewStack = new CardPanel(); //stack of mainApplet modules
21
22          public void init() {
23                  insets();
24                  setFont(f);
25                  setBackground((Color.lightGray).darker());
26                  setLayout(new BorderLayout(0,10));
27                  resize(600,510);
28                  /*north panel*/
29                          pn.setBackground(Color.lightGray);
30                          pn.setForeground(Color.blue);
31                          pn.add(title);
32                          pn.resize(this.size().width,10);
33                          add("North", pn);
34                  /*south panel*/
35                          ps.setBackground(Color.lightGray);
36                          ps.setForeground(Color.blue);
37                          ps.setLayout(new GridLayout(1,2,5,5));
38                          ps.add(saveCard);
39                          ps.add(switchCard);
40                          add("South", ps);
```

```
41                  /*center panel*/
42                          pc.setBackground(Color.white);
43                          pc.add(viewStack);
44                          add("Center",pc);
45                  // start with Entry Form
46                          labelCard("Entry Form for Creating Your Simulation Model",
47                                  "Save Model","View Simulation NOW");
48                  ((CardLayout)viewStack.getLayout()).show(viewStack,"Entry Form");
49          }
50
51          public Insets insets() {
52                  return new Insets(10,10,10,10);
53          }
54
55          private void labelCard(String moduleLabel, String saveLabel,
56                                  String buttonLabel) {
57                  title.setText(moduleLabel);
58                  saveCard.setLabel(saveLabel);
59                  switchCard.setLabel(buttonLabel);
60          }
61
62          public boolean action(Event evt, Object arg) {
63                  if (evt.target instanceof Button) {
64                          if ("View Simulation NOW".equals(arg)) {
65                                  viewStack.transferDataS();
66                                  labelCard("Interface for Viewing Your Simulation Model",
67                                          "Save Output","Define/ Revise Model");
68          ((CardLayout)viewStack.getLayout()).show(viewStack,"Simulator");
69                          }
70                          else if("Define/ Revise Model".equals(arg)) {
71                             labelCard("Entry Form for Creating Your Simulation Model",
72                                          "Save Model","View Simulation NOW");
73          ((CardLayout)viewStack.getLayout()).show(viewStack,"Entry Form");
74                          }
75                          else if("Save Model".equals(arg)) {
76                                  /* add functionality for saving model */
77                          }
78                          else if("Save Output".equals(arg)) {
79                                  /* add functionality for saving output */
80                          }
```

```
81                              return true;
82                      }
83                      return false;
84              }
85      } //endof MainApplet class
```

```
1   package netSim;
2   import java.util.*;
3
4   /**
5           ModelParser extends StringTokenizer
6           Parses textual information from the entry panel into forms compatible
7           with the databases.
8   */
9
10  class ModelParser extends StringTokenizer  {
11          String rule;
12          String delimiter;
13          String new_value = null;
14          int num_tokens = countTokens();
15
16          public ModelParser(String rl) { //conditions & rules
17                  super(rl, " ", false);
18                  rule=rl;
19          }
20
21          public ModelParser(String rl, String delim) { //rnd distr. & state vars
22                  super(rl, delim, false);
23                  rule = rl;
24                  delimiter=delim;
25          }
26
27          protected String[] decipherVars() {
28                  int i = 0;
29                  String[] splitVar = new String[(this.countTokens())*2];
30                  while (this.hasMoreTokens()) {
31                          String thisVar = nextToken();
32                          ModelParser varMP = new ModelParser(thisVar, "=");
33                          int num_tokens = varMP.countTokens();
34                          if (num_tokens-->0) {
35                                  splitVar[i] = varMP.nextToken();
36                                  splitVar[i+1] = varMP.nextToken();
37                                  i = i+2;
38                          }
39                  }
40                  return(splitVar);
```

```
41          }
42
43          protected int[] decipherRndParam() {
44                  int[] parameter_values = new int[2];
45                  int p0 =0,p1=0;
46                  while (this.hasMoreTokens()) {
47                          if (num_tokens == 2) {
48                                  p0 = Integer.parseInt(nextToken());
49                                  p1 = Integer.parseInt(nextToken());
50                          }
51                          else /*(num_tokens=1)*/ {
52                                  p0 = Integer.parseInt(nextToken());
53                                  p1 = 0;
54                          }
55                  }
56                  parameter_values[0] = p0;
57                  parameter_values[1] = p1;
58                  return(parameter_values);
59          }
60
61          protected boolean decipherCondition(MyHashtable ht) {
62                  MyHashtable actualDataValues=ht;
63                  boolean chk = false;
64                  char ch = '\0';
65                  int testValue = 0;
66                  try {
67                          new_value = nextToken();
68                  } catch (NoSuchElementException nsee1) {};
69                  //db("this Element is --> "+new_value); //debug
70                  if (new_value.equals("true") | (new_value.equals("TRUE"))) {
71                          chk = true;
72                  }
73                  else {
74                          for (Enumeration e = actualDataValues.elements(),
75                                  k = actualDataValues.keys(); e.hasMoreElements();){
76                                  Integer elem= (Integer) e.nextElement();
77                                  String key = (String) k.nextElement();
78                                  //db("key: "+key+" elem: "+elem); //debug
79                                  if (key.equals(new_value)) {
80                                          num_tokens = num_tokens-1;
```

```
                                        try {
                                                ch = nextToken().charAt(0);
                                        } catch (NoSuchElementException nsee2) {};
                                        //db("the character is " + ch); //debug
                                        num_tokens = num_tokens-1;
                                        try {
                                                testValue = Integer.parseInt(nextToken());
                                        } catch (NoSuchElementException nsee3) {};
                                        //db("the testValue is " + testValue); //debug
                                        switch (ch) {
                                        case '=' : {chk = tryEqual(elem, testValue); break;}
                                        case '>' : {chk = tryGreater(elem, testValue); break;}
                                        case '<' : {chk = tryLess(elem, testValue); break;}
                                        case '!' : {chk = tryNot(elem, testValue); break;}
                                        }
                                        break;
                                }
                        }
                }
                return(chk);
        }

        private boolean tryEqual(Integer isNow, int compareTo) {
                boolean ans = false;
                if (isNow.intValue() == compareTo) {
                        //db("was equal??: "+isNow.intValue()+" = "+compareTo); //debug
                        ans = true;
                }
                return(ans);
        }

        private boolean tryGreater(Integer isNow, int compareTo) {
                boolean ans = false;
                if (isNow.intValue() > compareTo) {
                     //db("was greater??: "+isNow.intValue()+" > "+compareTo); //debug
                    ans = true;
                }
                return(ans);
        }

```

```
121         private boolean tryLess(Integer isNow, int compareTo) {
122                 boolean ans = false;
123                 if (isNow.intValue() < compareTo) {
124                         //db("was less??: "+isNow.intValue()+" < "+compareTo); //debug
125                         ans = true;
126                 }
127                 return(ans);
128         }
129
130         private boolean tryNot(Integer isNow, int compareTo) {
131                 boolean ans = false;
132                 if (isNow.intValue() != compareTo) {
133                     //db("was not equal??: "+isNow.intValue()+" !="+compareTo); //debug
134                     ans = true;
135                 }
136                 return(ans);
137         }
138
139         protected Integer decipherRule(Integer old, Integer intTime,
140                         MyHashtable vd, MyVector W) {
141                 Integer orgValue=old;
142                 Integer calculatedValue = null;
143                 int increment = 0;
144                 String stringIndex = null;
145                 int intIndex = 0;
146                 while (this.hasMoreTokens()) {
147                         new_value = nextToken();
148                         //db("this Element is --> "+new_value); //debug
149                         //db("number of tokens now --> "+num_tokens); //debug
150                         if ((new_value.equals("+")) & (num_tokens >1))  {
151                                 num_tokens = num_tokens-1;
152                                 String next = (String) nextToken();
153                                 //db("number of tokens after '+' --> "+num_tokens); //debug
154                                   try{ increment = Integer.parseInt(next);      //treat as int
155                                         //db(" add increment --> "+increment); //debug
156                                 calculatedValue=increaseVar(orgValue,increment);
157                                         orgValue = calculatedValue;
158                                       } catch (NumberFormatException nfe) {
159                                             if (next.equals("clk")) {
160                                               increment = intTime.intValue();
```

```
161                     calculatedValue=increaseVar(orgValue,increment);
162                                       orgValue = calculatedValue;
163                                     }
164                                     else if (next.startsWith("W[")) {
165
166                 //parse out into vector
167                     stringIndex = next.substring(2,(next.length() -1));
168                     for (Enumeration e3 = vd.elements(), k3 = vd.keys();
169                             e3.hasMoreElements();){
170                             Integer elem3= (Integer) e3.nextElement();
171                             String key3 = (String) k3.nextElement();
172                             if (key3.equals(stringIndex)) {
173                                     intIndex = elem3.intValue();
174                                     break;
175                             }
176                     }
177                     Integer integerIndex =(Integer)W.elementAt(intIndex);
178                     increment = integerIndex.intValue();
179                     calculatedValue=increaseVar(orgValue,increment);
180                     orgValue = calculatedValue;
181                 } else  {
182                     for (Enumeration e = vd.elements(), k = vd.keys();
183                             e.hasMoreElements();){
184                              Integer elem= (Integer) e.nextElement();
185                              String key = (String) k.nextElement();
186                               if (key.equals(next)) {
187                                 increment = elem.intValue();
188                            calculatedValue=increaseVar(orgValue,increment);
189                                 orgValue = calculatedValue;
190                                 break;
191                               }
192                     }
193                 }
194                 }
195                 }
196                 else if ((new_value.equals("-")) & (num_tokens >1))  {
197                     num_tokens = num_tokens-1;
198                     String next = (String) nextToken();
199                     //db("number of tokens after '-' --> "+num_tokens); //debug
200                     try{ increment = 0-Integer.parseInt(next);
```

```
201                         //db(" minus increment --> "+increment); //debug
202                         calculatedValue=increaseVar(orgValue,increment);
203                         } catch (NumberFormatException nfe) {
204                                 if (next.equals("clk")) {
205                                         increment = 0-intTime.intValue();
206                         calculatedValue=increaseVar(orgValue,increment);
207                                         orgValue = calculatedValue;
208                                 }
209                                 else if (next.startsWith("W[")) {
210                         //parse out into vector
211                         stringIndex = next.substring(2,(next.length() -1));
212                         for (Enumeration e3 = vd.elements(), k3 = vd.keys();
213                                 e3.hasMoreElements();){
214                                 Integer elem3= (Integer) e3.nextElement();
215                                 String key3 = (String) k3.nextElement();
216                                 if (key3.equals(stringIndex)) {
217                                         intIndex = elem3.intValue();
218                                         break;
219                                 }
220                         }
221                         Integer integerIndex =(Integer)W.elementAt(intIndex);
222                         increment = 0-(integerIndex.intValue());
223                         calculatedValue=increaseVar(orgValue,increment);
224                         orgValue = calculatedValue;
225                 } else{
226                         for (Enumeration e = vd.elements(), k = vd.keys();
227                                 e.hasMoreElements();){
228                                 Integer elem= (Integer) e.nextElement();
229                                  String key = (String) k.nextElement();
230                                   if (key.equals(next)) {
231                                         increment = 0-elem.intValue();
232                                         calculatedValue=increaseVar(orgValue,increment);
233                                         orgValue = calculatedValue;
234                                         break;
235                                   }
236                         }
237                         }
238                         }
239                         }
240                         else if (new_value.equals("clk")) {
```

```
241                                     num_tokens = num_tokens-1;
242                                     //db("number of tokens after 'clk' --> "+num_tokens);
243                                     //System.out.print(" the time now --> "+increment); //debug
244                                     calculatedValue=intTime;
245                                     orgValue = intTime;
246                             }
247                             else {      //assume its an integer
248                                     //db(" int? --> "+new_value); //debug
249                                     //db("number of tokens with this int --> "+num_tokens);
250                                     increment =Integer.parseInt(new_value);
251                                     calculatedValue=replaceVar(increment);
252                             }
253                     }
254                     return(calculatedValue);
255             }
256
257             private Integer increaseVar(Integer isNow, int howMuch) {
258                     int v = isNow.intValue();
259                     int d = howMuch;
260                     Integer update = new Integer(v + d);
261                     //db("increaseVar:"+"v "+v+" d "+d+" update "+update); //debug
262                     return(update);
263             }
264
265             private Integer replaceVar(int howMuch) {
266                     Integer update = new Integer(howMuch);
267                     return(update);
268             }
269
270             public void db(String toDebug) {
271                     System.out.println(toDebug);
272             }
273     } //endof ModelParser class
```

```
1   package netSim;
2   import java.util.Hashtable;
3   import java.util.Enumeration;
4
5   /**
6           MyHashtable extends Hashtable
7           Extends java.util.Hashtable, purely for debugging purposes.
8   */
9
10  class MyHashtable extends Hashtable {
11          public MyHashtable(int size) {
12                  super(size);     // number of items per MyHashtable
13          }
14
15          public void listAll() {
16                  System.out.println("Key   ;  Element ");
17                  for (Enumeration e = elements(), k = keys(); e.hasMoreElements();) {
18                          System.out.println(k.nextElement() + " ;  "+  e.nextElement());
19                  }
20          }
21
22          public void listAllE() {
23                  for (Enumeration e = elements(), k = keys(); e.hasMoreElements();) {
24                          Object[] ky = (Object[]) k.nextElement();
25                          Object el = (Object) e.nextElement();
26                          System.out.print(ky[0]+" "+ky[1]+ ":");
27                          System.out.println(el.toString());
28                  }
29          }
30  } //endof MyHashtable class
```

```
1   package netSim;
2   import java.util.Vector;
3   import java.util.Enumeration;
4
5   /**
6            MyVector extends Vector
7           Extends java.util.Vector, purely for debugging purposes.
8   */
9
10  class MyVector extends Vector {
11          public MyVector(int size) {
12                  super(size);     // number of items per MyVector
13          }
14
15          public void listAll() {
16                  for (Enumeration e = elements(); e.hasMoreElements();) {
17                          Object[] el = (Object[]) e.nextElement();
18                          System.out.print(this.indexOf(el) +": ");
19                          for (int i = 0; i< el.length; i++) {
20                                  System.out.print(el[i]+", ");
21                          }
22                          System.out.println(" ");
23                  }
24          }
25
26          public void listAllC() {
27                  for (Enumeration e = elements(); e.hasMoreElements();) {
28                          char[] el = (char[]) e.nextElement();
29                          System.out.print(this.indexOf(el) +": ");
30                          for (int i = 0; i< el.length; i++) {
31                                  System.out.print(el[i]+", ");
32                          }
33                          System.out.println(" ");
34                  }
35          }
36  } //endof MyVector class
```

```
1   package netSim;
2   import java.util.Random;
3
4   /**
5           RndGenerator extends Random
6           Determines the random number stream for the current simulation run and
7           calculates random variates from that stream as needed.
8   */
9
10  class RndGenerator extends Random  {
11          double aRND,aLnNum;
12          //double JaLnNum = 0, JthisRndNum = 0;  //debugging vars
13          int i;
14          long thisSeed = 0;
15
16          public RndGenerator() {
17                  super();
18          }
19
20          public RndGenerator(long theSeed) {
21                  super(theSeed);
22                  thisSeed = theSeed;
23          }
24
25          protected Double calcRndNum(String description) {
26                  double thisRndNum=0;
27                  int first =0, second=0;
28                  if ((!description.equals(""))&(!description.equals("0"))) {
29                          String p = description.substring(4,(description.length() -1));
30                          ModelParser rg= new ModelParser(p, ",");
31                          int[] parameters = rg.decipherRndParam();
32                          first = parameters[0];
33                          second = parameters[1];
34                          // use sigma numbers or not
35                          if (thisSeed==-2) {
36                                  //aRND = sigmaRndNum();
37                                  aLnNum = sigmaNaturalLogOfRndNum();
38                          }
39                          else {
40                                  aRND = this.nextDouble();
```

```
41                                  aLnNum = Math.log(aRND);
42                          }
43                          if (description.startsWith("exp")) {
44                                  thisRndNum = -(first)*(aLnNum);
45                                  //JaLnNum = Math.log(aRND); //debugging
46                                  //JthisRndNum = -(first)*(JaLnNum);  //debugging
47                                  //System.out.println("aLnNum:"+aLnNum+"
48                                  // thisRndNum:"+thisRndNum);
49                                  //System.out.print("Java calculated:
50  JaLnNum:"+JaLnNum);
51                                  //System.out.println(" & this RNV:"+JthisRndNum);
52                          }
53                          else if (description.startsWith("uni")) {
54                                  thisRndNum = (aRND*(second-first))+first;
55                          }
56                          else if (description.startsWith("sta")) {
57                                  thisRndNum =first;
58                          }
59                  }
60                  Double thisDblRndNum = new Double(thisRndNum);
61
62                  return(thisDblRndNum);
63          }
64
65          private double sigmaRndNum() {
66                  //double[] list={0.096,0.754,0.236,0.246, 0.740, 0.583, 0.095, 0.336,
67                  //                                  0.669, 0.326, 0.017, 0.831, 0.083, 0.034};
68                  double[] list=
69          {96671.312,754759.125,236939.765,246429.515,740916.5,583869.5,
70          95099.476,336982.75,669204.875,326650.656,17661.154,831027.812,
71          83955.187,34709.234,358116.062,856788.125,37192.121,87996.078,
72          950118.125,635317.812,786756.5,16543.078,39492.566,751582.875,
73          853431.562,624260.75,950330.625,207022.312,424055.812,106222.578,
74          282898.218,670186.25,819377.75,281673.312,83438.398,349161.156,
75          351590.531,182322.468,293786.093,663260.812,424314.937,461135.312,
76          301270.375,451273.718,557435.312,816026.187,952038,902452.187,
77          513509.437,553682.625,743496.312,942761.375,989804.812,649291.437,
78          641024.437,696939.5,462714.968,850451.5,538471.5,90573.867,
79          274996.562,866795.875,238334.906,694889.25,3367.645,600012.875,
80          416953.218,732717.562,784015.5,948577.062,734034.812,923812.375,
```

```
514397.156,473043.343,439806.906,834829.812,984389.687,637366.812};
        double nextRnd = (list[i]);
        i = i+1;
        return(nextRnd);
    }

    private double sigmaNaturalLogOfRndNum() {
        //double[] list={-2.336,-0.281,-1.439,-1.4,-0.299,-0.538,-2.352,-1.087,
        //                    -0.401,-1.118,-4.036,-0.185,-2.477,-3.36};
        double[] list=
    {-2336438.8,-281356.59,-1439949.4,-1400679.3,-299867.38,-538077.75,
    -2352831.8,-1087723.5,-401665,-1118864,-4036388,-185092.02,
    -2477472,-3360749.5,-1026898.1,-154564.66,-3291658.5,-2430463,
    -51168.96,-453629.94,-239836.45,-4101787.5,-3231642.8,-285573.81,
    -158489.89,-471187.19,-50945.335,-1574928.8,-857890.19,-2242218.5,
    -1262668.1,-400199.66,-199210.11,-1267007.4,-2483646.8,-1052221.8,
    -1045287.9,-1701978.4,-1224903.4,-410586.97,-857279.31,-774063.69,
    -1199747.3,-795681.25,-584408.75,-203308.81,-49150.339,-102639.59,
    -666486.88,-591163.63,-296391.5,-58942.089,-10247.523,-431873.59,
    -444687.75,-361056.63,-770644,-161987.86,-619020.69,-2401589.8,
    -1290996.8,-142951.73,-1434078.5,-364002.78,-5693541.5,-510804.13,
    -874781.25,-310994.94,-243326.52,-52792.261,-309198.78,-79246.265,
    -664759.63,-748568.25,-821419.5,-180527.41,-15733.461,-450409.91};
        double nextLnNum = (list[i]);
        i = i+1;
        return(nextLnNum);
    }
}// end of RndGenerator class
```

```
1   package netSim;
2   import java.util.*;
3   import java.awt.*;
4
5   /**
6           ViewerPanel extends Panel implements Runnable
7           This file contains three classes: ViewerPanel, SchedThread, AnimationCanvas.
8
9           ViewerPanel defines and draws the GUI for viewing and interacting with the
10          simulation model.
11  */
12
13  public class ViewerPanel extends Panel implements Runnable {
14          boolean wasPaused = false;
15          DataDictionary DD = new DataDictionary();
16          Thread viewerThread;
17          SchedThread simThread;
18          String[] startNode;
19
20          Panel pn = new Panel();                         //North
21          Panel pn1 = new Panel();                //North -left
22          Button playpauseButton = new Button("PLAY");
23          Button stopButton = new Button("STOP");
24          Choice c = new Choice();                //North-middle
25          Panel pn2 = new Panel();                //North - right
26          TextField sd = new TextField(7); //seed field
27          TextField rl = new TextField(7); //runlength field
28          Panel pc = new Panel();                         //Center
29          AnimationCanvas cc1 = new AnimationCanvas(DD,this,550,130);
30                  //Center top - holds animation
31          Panel pc2  = new Panel();       //Center bottom - holds note
32          Label note = new Label("",Label.LEFT);
33          TextArea t = new TextArea(6,50);    //***South -- data output
34          MyHashtable initEvent;
35
36          public ViewerPanel() {
37                  t = new TextArea(6,50);
38          }
39
40          protected void putData(MyHashtable htEV, MyHashtable ht1, MyHashtable
```

```
41    ht2,
42                  MyHashtable ht3, MyHashtable ht4, MyHashtable ht5, MyHashtable
43    ht6,
44                        MyHashtable htED, MyVector v1, MyVector v2, MyVector v3,
45                        MyVector v4, MyVector v5, MyVector v6){
46                  DD.putData(htEV, ht1, ht2, ht3, ht4, ht5, ht6, htED, v1,v2,v3,v4,v5,v6);
47            }
48
49            protected void initModel(String[] st, MyHashtable ie) {
50                  startNode=st;
51                  initEvent=ie;
52                  DD.initData(initEvent);
53            }
54
55            public void start() {
56                  if (viewerThread == null) {
57                        viewerThread = new Thread(this);
58                        viewerThread.start();
59                  }
60            }
61
62            public void run() {
63                  setBackground((Color.lightGray).darker());
64                  insets();
65                  setLayout(new BorderLayout(0,10));
66                  resize(570,400);
67
68                  /*north panel*/
69                  pn.setBackground(Color.lightGray);
70                  pn.setLayout(new GridLayout(1,2,5,5));
71                  pn.resize(this.size().width, 40);
72                              c.addItem("Animation & Data");
73                              c.addItem("Animation only");
74                              c.addItem("Data Output only");
75                        pn1.setLayout(new FlowLayout(FlowLayout.CENTER,5,5));
76                        pn1.add(playpauseButton);
77                        pn1.add(stopButton);
78                        pn1.add(c);
79                              sd.setBackground(Color.white);
80                              setSD();
```

```
81                              rl.setBackground(Color.white);
82                              setRL();
83                      pn2.setLayout(new GridLayout(2,2));
84                      pn2.add(new Label("random number seed:"));
85                      pn2.add(sd);
86                      pn2.add(new Label("model run-length:"));
87                      pn2.add(rl);
88              pn.add(pn1);
89              pn.add(pn2);
90              add("North", pn);
91
92              /*south panel*/
93              t.setBackground(Color.lightGray);
94              t.appendText("Data output:");
95              add("South", t);
96
97              /*center panel*/
98              pc.setLayout(new BorderLayout(0,10));
99                      note.resize(this.size().width,20);
100                     note.setAlignment(Label.LEFT);
101                     pc2.setBackground(Color.white);
102                     pc2.setForeground(Color.magenta);
103                     pc2.resize(this.size().width,20);
104                     pc2.add(note);
105             pc.add("South",pc2);
106                     cc1.setBackground(Color.white);
107                     cc1.resize(550,130); // sets canvas size
108             pc.add("Center",cc1);
109             add("Center", pc);
110             stop();
111     }
112
113     public Insets insets() {
114             return new Insets(10,10,10,10);
115     }
116
117     public boolean action(Event evt,Object arg){
118             if ((evt.target instanceof Button) && ("STOP".equals(arg))){
119                     stopB();
120                     return true;
```

```
121                     }
122                     else if ((evt.target instanceof Button) && ("PAUSE".equals(arg))){
123                             pauseB();
124                             return true;
125                     }
126                     else if ((evt.target instanceof Button) && ("PLAY".equals(arg))){
127                             playB();
128                             return true;
129                     }
130                     return false;
131             }
132
133             protected void stopN(){                 //called from SchedThread to end run
134                             note.resize(this.size().width,20);
135                             note.setText("SIMULATION FINISHED");
136                             simThread.stop();
137                             simThread = null;
138             }
139
140             private void stopB(){                   //user stop sim by button
141                     if (simThread !=null) {
142                             playpauseButton.setLabel("PLAY");
143                             note.resize(this.size().width,20);
144                             note.setText("SIMULATION STOPPED: press PLAY to
145     restart.");
146                             simThread.stop();
147                             simThread = null;
148                             t.appendText("\n  ** Simulation run aborted. ** \n");
149                     }
150             }
151
152             private void pauseB() {                         //user pause by button
153                     if (simThread.isAlive()) {              //simulation is running
154                             note.resize(this.size().width,20);
155                             note.setText("SIMULATION PAUSED: press PLAY to
156     resume.");
157                             playpauseButton.setLabel("PLAY");
158                             simThread.suspend();            //pause sim until play button
159                             wasPaused = true;
160                     }
```

```
161         }
162
163         private void playB() {                          //user play,start by button
164                 note.resize(this.size().width,20);
165                 note.setText("SIMULATION IN PROGRESS");
166                 playpauseButton.setLabel("PAUSE");
167                 if (simThread==null) {                  //sim was stopped
168                         long valueSD = setSD();
169                         double valueRL = setRL();
170                         simThread = new SchedThread(DD,this,valueSD,valueRL);
171                         simThread.start();
172                 }
173                 else  if (wasPaused){           //sim was paused
174                         playpauseButton.setLabel("PAUSE");
175                         wasPaused=false;
176                         simThread.resume();
177                 }
178         }
179
180         private long setSD() {
181                 long newSd;
182                 newSd = readSeed(sd.getText());
183                 Long newSeed =new Long(newSd);
184                 sd.setText(newSeed.toString());
185                 return(newSd);
186         }
187
188         private double setRL() {
189                 double newRL;
190                 newRL=readRunLength(rl.getText());
191                 Double newRun =new Double(newRL);
192                 rl.setText(newRun.toString());
193                 return(newRL);
194         }
195
196         private long readSeed(String seedChangeTo) {
197                 long theSeed = 0;
198                 if ((seedChangeTo.equals(""))|(seedChangeTo == null)) {
199                         theSeed = System.currentTimeMillis();
200                 }
```

```
201                     else {
202                             try {
203                                     theSeed = (Long.valueOf(seedChangeTo)).longValue();
204                             }
205                             catch (NumberFormatException e) {};
206                     }
207                     return(theSeed);
208             }
209
210             private double readRunLength(String lengthChangeTo) {
211                     double runLength = 0;
212                     if ((lengthChangeTo.equals(""))|(lengthChangeTo == null)) {
213                             runLength = 1440;
214                     }
215                     else {
216                             try {
217                             runLength = (Double.valueOf(lengthChangeTo)).doubleValue();
218                             }
219                             catch (NumberFormatException e) {};
220                     }
221                     return(runLength);
222             }
223
224             public void stop() {
225                     if (viewerThread != null) {
226                             viewerThread.stop();
227                             viewerThread = null;
228                     }
229             }
230     } //endof ViewerPanel class
231
```

```
232  class SchedThread extends Thread {
233          double runLength;
234          double time;
235          Double nextTime = null;
236          String[] currentEvent;
237          Object[] nextListItem;
238          MyHashtable schedEvent;
239
240          DataDictionary processDD;
241          ViewerPanel viewerVP;
242
243  /**
244          SchedThread extends Thread
245          Monitors the simulation clock and future event list, determines the type of
246          output desired by the user, notifies the database of the current random
247          number seed, and signals the database to process the next event.  This class
248          manages the simulation run until the clock exceeds the current run-length,
249          the future events list is empty, or the simulation is stopped by the user.
250
251          SchedThread must be contained in same text file as ViewerPanel.
252  */
253
254          public SchedThread(DataDictionary data, ViewerPanel target,
255                          long sd, double rl) {
256                  super();
257                  processDD=data;
258                  viewerVP=target;
259                  time = 0;
260                  runLength = rl;
261                  processDD.setUp(sd, viewerVP.initEvent);
262                  currentEvent= target.startNode; //set init node
263                  setPriority(Thread.MIN_PRIORITY);
264          }
265
266          public void run(){
267                  if ((viewerVP.c.getSelectedIndex()==0)
268                                  |(viewerVP.c.getSelectedIndex()==2)){
269                  //(animation&data output)|(data output))
270                  viewerVP.t.appendText("\n time; event; Queue Length; Throughput;");
271                  viewerVP.t.appendText(" Waiting Time; Server Busy Time"); //outputdata
```

```
272                         //System.out.println("time; event; Q; CO; WT; TE"); //output
273 quick
274                 }
275                 while (time<=runLength) {
276                         //System.out.print("currentEvent ");
277                         //System.out.println(currentEvent[0]+", "+currentEvent[1]);
278                         schedEvent = (MyHashtable)
279 processDD.events.get(currentEvent);
280                         //System.out.println("schedEvent ");
281                         //schedEvent.listAll();
282                         processDD.updateEventVars(schedEvent, time);
283                         processDD.updateEventSch(currentEvent, time);
284                         nextListItem = processDD.adjustEventSch();
285                         if ((viewerVP.c.getSelectedIndex()==0)
286                                         |(viewerVP.c.getSelectedIndex()==2)){
287                                 //(animation&data output)|(data output))
288                                         outputData();
289                         }
290                         if ((viewerVP.c.getSelectedIndex()==0)
291                                         |(viewerVP.c.getSelectedIndex()==1)){
292                                 //(animation&data output)|(animation))
293                                         viewerVP.cc1.repaint(currentEvent, nextListItem);
294                                         try { Thread.sleep(910); }
295                                         catch (InterruptedException e) { };
296                         }
297                         currentEvent = (String[]) nextListItem[2];
298                         nextTime = (Double) nextListItem[0];
299                         time = nextTime.doubleValue();
300                                 //System.out.println("time: "+time);//debug
301                                 //System.out.print("nextListItem:  ");
302                                 //System.out.println(nextListItem[0]+ ",
303 "+nextListItem[1]+
304                                 // ",  {"+currentEvent[0]+", "+currentEvent[1]+"}");
305                 }
306                 //System.out.println("==================");
307                 viewerVP.t.appendText("\n ============================");
308                 viewerVP.t.appendText("\n Simulation run has finished.");
309                 viewerVP.t.appendText("\n ============================");
310                 viewerVP.playpauseButton.setLabel("PLAY");
311                 viewerVP.stopN();
```

```
312                     this.stop();
313             } //endof run
314
315             public void outputData() {
316                     Integer VI1= (Integer) processDD.varData.get("Q");
317                     double v1= VI1.doubleValue()/10000;
318                     Integer VI2= (Integer) processDD.varData.get("CO");
319                     double v2= VI2.doubleValue()/10000;
320                     Integer VI3= (Integer) processDD.varData.get("WT");
321                     double v3= VI3.doubleValue()/10000;
322                     Integer VI4= (Integer) processDD.varData.get("TE");
323                     double v4= VI4.doubleValue()/10000;
324                     viewerVP.t.appendText("\n "+time +";  "+currentEvent[1]+";  ");
325                     viewerVP.t.appendText(v1+";  ");
326                     viewerVP.t.appendText(v2+";  ");
327                     viewerVP.t.appendText(v3+";  ");
328                     viewerVP.t.appendText(v4+" ");
329                     //System.out.println(time +";  "+currentEvent[1]+";  "
330                     //+processDD.varData.get("Q")+";  "+processDD.varData.get("CO")+";  "
331                     //+processDD.varData.get("WT")+";  "+processDD.varData.get("TE"));
332                     //processDD.W.listAll();
333             }
334     }//endof SchedThread class
```

```
335  class AnimationCanvas extends Canvas{
336          int activeIndex;
337          int nextIndex;
338          DataDictionary processDD;
339          MyVector eVectList = new MyVector(10);
340          ViewerPanel viewerVP;
341
342          static int xinit = 20, y1 = 35;                    //init point for nodes
343          static int xwidth = 40, ywidth = 40;   //size of nodes
344          char[] elabel;                                                    //node labels
345          int x1,x2;
346          int xlabel, ylabel;
347          int[] Exes, Whys;
348          int Pts;
349
350          Image offscreenImg;
351          Graphics offscreenG;
352          Font f = new Font("TimesRoman",Font.PLAIN,10);
353          Dimension imageSize;
354
355  /**
356           AnimationCanvas extends Canvas
357          Draws the event graph of the simulation model onto the viewing panel.
358          Animates the event graph by periodically repainting sections as requested
359          by the simulation thread, an instance of SchedThread.
360
361          AnimationCanvas must be contained in same text file as ViewerPanel.
362  */
363
364          public AnimationCanvas(DataDictionary data, ViewerPanel target,
365                                  int width,int height) {
366                  processDD = data;
367                  this.viewerVP = target;
368                  imageSize = new Dimension(width,height);
369          }
370
371          public void paint(Graphics g) {
372                  offscreenImg = createImage(imageSize.width, imageSize.height);
373                  offscreenG = offscreenImg.getGraphics();
374                  offscreenG.setFont(f);
```

```
375                 eVectList.removeAllElements();
376                 makeNodeList();
377                 //System.out.println("ready to draw nodes"); //debug
378                 for (int i=0; i < eVectList.size();i++) {
379                         locateNode(i);
380                         offscreenG.setColor(Color.yellow);
381                         drawNode(g);
382                 }
383                 //System.out.println("starting edges");  //debug
384                 for (Enumeration e2 = processDD.edges.elements(),
385                                 k2 = processDD.edges.keys(); e2.hasMoreElements();){
386                         MyVector edgeList= (MyVector)e2.nextElement();
387                         String[] thisKey2 = (String[])k2.nextElement();
388                         int edgeIndexFrom = Integer.parseInt(thisKey2[0]);
389                         char[] edgeNameFrom = thisKey2[1].toCharArray();
390                         //System.out.println("edgeFirstName "+edgeNameFrom[0]+" "
391                         //                +edgeNameFrom[1]+" "+edgeNameFrom[2]);
392                         for (int dest = 0; dest<edgeList.size(); dest = dest+4) {
393                                 String[] thisElem = (String[])edgeList.elementAt(dest);
394                                 int edgeIndexTo = Integer.parseInt(thisElem[0]);
395                                 for (Enumeration e3 = eVectList.elements();
396                                   e3.hasMoreElements();) {
397                                   char[] currentNode= (char[]) e3.nextElement();
398                                   if (edgeIndexFrom == eVectList.indexOf(currentNode)) {
399         whichEdge(edgeIndexTo, eVectList.indexOf(currentNode), edgeIndexFrom);
400                                                 offscreenG.setColor(Color.yellow);
401                                                 drawArrow(g);
402                                                 break;
403                                         }
404                                 }
405                         }
406                 }
407                 this.viewerVP.cc1.getGraphics().drawImage(offscreenImg,0,0,this);
408         } //end of paint
409
410         public void repaint(String[] event, Object[] item) {
411                 activeIndex = Integer.parseInt(event[0]);
412                 String[] nextNode = (String[]) item[2];
413                 nextIndex = Integer.parseInt(nextNode[0]);
414                 repaint(0);
```

```
415         }
416
417         public void update(Graphics g) {
418                 //System.out.println("start update method");
419                 for (Enumeration e3 = eVectList.elements(); e3.hasMoreElements();) {
420                         char[] listNode= (char[]) e3.nextElement();
421                         int listIndex = eVectList.indexOf(listNode);
422                         //System.out.println("listIndex "+listIndex);
423                         if (activeIndex == listIndex) {
424                                 /*redraw node*/
425                                 locateNode(activeIndex);
426                                 offscreenG.setColor(Color.cyan);
427                                 drawNode(g);
428                 this.viewerVP.cc1.getGraphics().drawImage(offscreenImg,0,0,this);
429                                 /*pause before repainting*/
430                                 try { Thread.sleep(220); }
431                                 catch (InterruptedException ie1) { };
432                                 offscreenG.setColor(Color.yellow);
433                                 drawNode(g);
434                 this.viewerVP.cc1.getGraphics().drawImage(offscreenImg,0,0,this);
435                                 /*check edges*/
436                                 for (Enumeration e2 = processDD.edges.elements(),
437                                                 k2 = processDD.edges.keys();
438                                                 e2.hasMoreElements();){
439                                         MyVector edgeList= (MyVector) e2.nextElement();
440                                         String[] thisKey2 = (String[])k2.nextElement();
441                                         int edgeIndexFrom = Integer.parseInt(thisKey2[0]);
442                                         if (edgeIndexFrom == activeIndex) {
443
444                         for (int dest = 0; dest<edgeList.size();dest =dest+4) {
445                                          String[] thisElem = (String[])edgeList.elementAt(dest);
446                                                 int edgeIndexTo = Integer.parseInt(thisElem[0]);
447                                                 if (edgeIndexTo == nextIndex) {
448                                                         /*redraw edges*/
449                                         whichEdge(nextIndex, listIndex, activeIndex);
450                                                 offscreenG.setColor(Color.cyan);
451                                                 drawArrow(g);
452                 this.viewerVP.cc1.getGraphics().drawImage(offscreenImg,0,0,this);
453                                                 /*pause before repainting*/
454                                                 try { Thread.sleep(450); }
```

```
455                                         catch (InterruptedException ie2) { };
456                                         offscreenG.setColor(Color.yellow);
457                                         drawArrow(g);
458             this.viewerVP.cc1.getGraphics().drawImage(offscreenImg,0,0,this);
459                                                         break;
460                                           }
461                                         }
462                                         break;
463                                     }
464                                 }
465                                 break;
466                         }
467             }
468             //pause before repainting
469             try { Thread.sleep(220); }
470             catch (InterruptedException ie3) { };
471     } //end of update
472
473     private void whichEdge(int n, int l, int a) {
474             int next = n;
475             int list = l;
476             int active = a;
477             if (next < list) {
478                     //System.out.println("call backwardEdge");
479                     backwardEdge(active);
480             }
481             else if (next == list) {
482                     //System.out.println("call curvedEdge");
483                     curvedEdge(active);
484             }
485             else if (next > list) {
486                     //System.out.println("call forwardEdge");
487                     forwardEdge(active);
488             }
489     }
490
491     private void makeNodeList() {
492             char[] eCharList = null;
493             String[] thisKey = null;
494             for (int j=0; j<processDD.events.size(); j++) {
```

```
495                         for (Enumeration k = processDD.events.keys();
496                                 k.hasMoreElements();){
497                                 thisKey = (String[])k.nextElement();
498                                 if (Integer.parseInt(thisKey[0]) == j) {
499                                         eCharList = (thisKey[1]).toCharArray();
500                                         eVectList.addElement(eCharList);
501                                         break;
502                                 }
503                         }
504                 }
505                 //System.out.println("eVectList: ");  //debug
506                 //eVectList.listAllC();  //debug
507         }
508
509         private void locateNode(int nodeIndex) {
510                 elabel = (char[]) eVectList.elementAt(nodeIndex);
511                 x1 = xinit+ nodeIndex*90;
512                 xlabel=x1+6;
513                 ylabel=y1+24;
514         }
515
516         private void drawNode(Graphics g) {
517                 offscreenG.fillOval(x1,y1,xwidth,ywidth);
518                 offscreenG.setColor(Color.black);
519                 offscreenG.drawOval(x1,y1,xwidth,ywidth);
520                 offscreenG.drawChars(elabel,0,elabel.length,xlabel,ylabel);
521         }
522
523         private void drawArrow(Graphics g) {
524                 offscreenG.fillPolygon(Exes,Whys,Pts);
525                 offscreenG.setColor(Color.black);
526                 offscreenG.drawPolygon(Exes,Whys,Pts);
527         }
528
529         private void backwardEdge(int startIndex) {
530                 x2=xinit+ startIndex*90;
531                 x1=x2 - 90;
532                 int[] exes = {x2,x1+47,x1+47,x1+40,x1+47,x1+47,x2};
533                 int[] whys = {y1+16,y1+16,y1+18,y1+15,y1+12,y1+14,y1+14};
534                 int pts = exes.length;
```

```
535                     Exes = exes;
536                     Whys = whys;
537                     Pts = pts;
538             }
539
540             private void curvedEdge(int startIndex) {
541               x1=xinit+ startIndex*90;
542           int[] exes = {x1+31,x1+31,x1+7,x1+7,x1+9,x1+6,x1+3,x1+5,x1+5,x1+33,x1+33};
543               int[] whys = {y1+4,y1-9,y1-9,y1-1,y1-1,y1+5,y1-1,y1-1,y1-11,y1-11,y1+6};
544               int pts = exes.length;
545              Exes = exes;
546               Whys = whys;
547               Pts = pts;
548             }
549
550             private void forwardEdge(int startIndex) {
551                     x1=xinit+ startIndex*90;
552                     x2=x1 + 90;
553                     int[] exes = {x1+40,x2-7,x2-7,x2,x2-7,x2-7,x1+40};
554                     int[] whys = {y1+21,y1+21,y1+23,y1+20,y1+17,y1+19,y1+19};
555                     int pts = exes.length;
556                     Exes = exes;
557                     Whys = whys;
558                     Pts = pts;
559             }
560     }//endof AnimationCanvas class
```

Appendix I: Netsim User's Manual

CONTENTS:

1. Use Requirements:

Netsim runs as a Java applet on any Java-compatible WWW browser or applet viewer.

The Netsim simulation package supports discrete-event simulation using event-graph modeling. While knowledge of simulation, particularly including the event graph approach, is useful in designing a model in Netsim, no knowledge of Java or simulation modeling is required to enter or modify the model.

To use Netsim one does need working knowledge of WWW browsers and must follow the formatting requirements stated in this user's manual.

2. Interfaces

View the simulation by clicking on the lower right button of the screen, "View Simulation NOW."

View the model parameters by clicking on the lower right button of the screen, "Define/ Revise Model."

3. Saving

The save buttons on the lower left side of the screen are currently disabled. Ideally, they would allow the user to name and save either the model specifications or the data output into a directory on the local computer.

4. Creating a Model

Switch to the entry form, if necessary, by clicking on the lower right button of the screen, "Define/ Revise Model."

All names and variables

- are case-sensitive,
- begin with a letter,
- do not contain spaces,
- appear in model in order listed.

4a. Events & Variables

Events/ Nodes

Netsim currently allows up to six events.

Enter the name of each event into the "event" text field on the entry form.

The events will appear on the animated model from left to right in the order they appear on the entry form.

Variables/ Attributes

Each event may contain as many variables and variable rules as desired.

Enter the variable rules for each event in the "state vars" text field following the associated event text field on the entry form.

- There are 2 reserved variables:
  clk , the current simulation time.
  W[ ], an array containing *clk* times and indexed by a given variable.
- All other variables are integer-valued.
- Separate variable name and rule by **=** .
- Separate equations by **;**.
- When used, place W[] last in the equation.
- *Note:*
  Leave no spaces between equation sections or equations.
  Within a variable rule leave one space between the operation and the value, as shown below.

The state variables will be processed in the order listed. This may make a difference in the results of a model if one variable references another during the same event.

Variable rules may do the following:		**Example:**
Change the existing value by	an integer, or the clock time.	Q=1 T=clk
Increase or decrease the existing value by	an integer, another variable, or the clock value.	Q=+ 1 Q=- TS T=+ clk
Create an array, W[] of clock times indexed by another variable.		W[Q]=clk
Combine any of these operations.		TE=clk - TS T=+ clk - W[Q])

4b. Edges

Location

Edges may be created between any two consecutive events, or an edge may be self-scheduling (i.e., from an event back to the same event). Netsim currently allows one edge in each direction between two different events, with up to one condition on each edge.

Type the name of the event where the edge begins in the "from" text box.

Type the name of the event where the edge ends in the "to" text box.

Note:
These names are case-sensitive and must match the event names defined in the event section of the entry form or the edge will not appear in the model.

Condition

Place basic conditions on the edges by using the format shown in the table below.

- The variable name may be any variable defined as a state variable in the upper section of the entry form.
- The integer value may be any integer.
- The operator is <, >, or =.
- The reserved word "TRUE" typed without the quotations makes that edge unconditional.
- *Note:*
  One space separates the operator from each of the other terms.

	var. name	<space>	operator	<space>	integer value
Examples:	Q		<		1
	Q		>		-1
	Q		=		0
			TRUE		

Time Delay

Netsim currently supports three types of time delays, as shown below.

Type the appropriate function in the "time delay" text box.

- Parameters of the function are
  integer-valued,
  separated by commas,
  enclosed by parentheses.
- *Note:*
  These time delay functions are case-sensitive and contain no spaces.

Time Delay:	Function:	Example:
constant increases	sta(a)	sta(5)
uniform random variates	uni(a,b)	uni(3,5)
exponential random variates	exp(a)	exp(5)

Priority

Type the preferred priority, integers 1 to 9, in the "priority" text box.

- 1 = lowest priority
- 5 = default setting
- 9 = highest priority

5. Viewing a Model

Switch to the output interface, if necessary, by clicking on the lower right button of the screen,

"View Simulation NOW."

To set the RNG seed:

Type any integer into the "random number seed" text box.

To set the simulation run length:

Type any integer into the "model run-length" text box.
Use the same units as in the time delay functions.

To change output type:

Hold the mouse button down over the "Animation & Data" box and select one of

- "Animation & Data,"
- "Animation only,"
- "Data Output only."

When selected, the model animates throughout the run. Current events and edges are displayed in blue.

The large scrollable text area displays data for selected variables at each event time. Variable selection for this display is not current available for the user.

To run simulation:

Click the mouse on the labeled buttons

- PLAY - starts or resumes a simulation run,
- PAUSE - pauses a run,
- STOP - aborts the run.

The PLAY/ PAUSE button is a toggle switch.

Note:
If the run finishes naturally, you must press STOP before starting another run. This is a bug in the current version.

Vita

Tamie Lynne Veith will receive her M.S. in Industrial and Systems Engineering at Virginia Polytechnic Institute and State University in May 1997. Her concentration is in Operations Research with particular interest in simulation and WWW applications. For the three years preceeding her graduate work, Tamie worked as a Technical Coordinator and Data Control Technician for the Sleep and Aging Research Program at the University of Washington in Seattle. She recieved her B.A. in Mathematics in 1992 from Reed College, Portland, Oregon.