AN ABSTRACT OF THE THESIS OF

Kurt Colvin for the degree of Master of Science in Industrial Engineering presented on May 21, 1997. Title: Development of a Discrete-Event, Object-Oriented Framework for Network-Centric Simulation Modeling Using Java.

Abstract Approved: Redacted for Privacy
Terrence G. Beaumariage

The primary objective of this research is to develop a network-centric simulation modeling framework that can be used to build simulation models through the use of Internet-based resources. An object-oriented programming approach was used to build a Java-based modeling framework focused on modeling a semiconductor fabrication system.

This research is an initial step in what may be a new network-centric simulation modeling methodology, where simulation models are created using software objects that are physically located in many different sites across the Internet. Once the ability to create and run a relatively simple model using a network-centric approach has been established, future research may lead to a simulation environment that not only lets a user interactively build models but also allows concurrent model development between a group of users, independent of their location, operating system, or computer architecture.

The prototype system implemented as a portion of this research is performed in the Java object-oriented programming language. A target system model is presented as an example of how the environment can be used to apply the network-centric simulation modeling methodology.

DEVELOPMENT OF A DISCRETE-EVENT, OBJECT-ORIENTED FRAMEWORK FOR NETWORK-CENTRIC SIMULATION MODELING USING JAVA

by

Kurt Colvin

A THESIS

Submitted to

Oregon State University

in partial fulfillment
of the requirements for the
degree of

Master of Science

Completed May 14, 1997
Commencement June 1998

Master of Science thesis of Kurt Colvin presented on May 21, 1997

APPROVED:

Redacted for Privacy

Major Professor, representing Industrial Engineering

Redacted for Privacy

Chair of Department of Industrial and Manufacturing Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University Libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Kurt Colvin, Author

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

EVELOPMENT OF A DISCRETE-EVENT, OBJECT-ORIENTED FRAMEWORK FOR NETWORK-CENTRIC SIMULATION MODELING USING JAVA

CHAPTER 1. INTRODUCITON

The primary objective of this research is to develop a network-centric simulation modeling framework that can be used to build simulation models through the use of Internet-based resources. An Object-Oriented Programming (OOP) approach was used to build a Java-based modeling framework focused on modeling a semiconductor fabrication system.

This research topic was chosen to further develop a new area of simulation called "Web-based simulation," to disseminate the benefits of simulation modeling, to further extend the functionality of Internet technologies, and to further develop the author's skills in simulation modeling, OOP and Internet application development.

A review of the literature indicated that much has been done in the area of simulation using OOP. OOP has proven to be a superior simulation methodology over traditional structural methodologies. One problem that has arisen is that the existing OOP models are available only to users of a specific programming language or simulation software platform. This research, however, concentrates on creating simulation models that are platform independent. In other words, the simulation model is available to virtually all users, regardless of their computer's hardware, software, configuration or location.

This research is an initial step in what may be a new network-centric modeling methodology, where simulation models are created using software objects that are physically located in many different sites across the Internet. Once the ability to create and run a relatively simple model using a network-centric approach has been established, the next step may be a simulation environment that not only lets a user interactively build models but also allows concurrent model development between a group of users, independent of their location, operating system, or computer architecture.

1.1 Statement of the Problem

Increasingly complex software applications continue to overload computer users. The area of simulation software is no different. As more sophisticated simulation studies are performed, more complex software systems are being developed to handle model requirements.

If a user wants to run a simulation model, the simulation software must first be installed and configured before the first simulation run can take place. With all of the possible computer configurations (hardware, operating system and software applications), the configuration and maintenance of simulation software becomes a significant task. Not only is the user tasked with complex configuration and maintenance of the local computing environment, but the simulation software developer must maintain multiple versions of software for each specific computer architecture.

In addition, the building of simulation models has traditionally been a "desktop-centric" task. In other words, the simulation models are created on a single computer at a specific geographic location, where the simulation software has been installed and configured. Once the model has been completed it can, of course, be distributed by media such as floppy diskettes or computer tape, or even more recently, transferred across the Internet via FTP or a similar method. However, there exists no method for concurrently developing model components in different locations, linking them together, and executing the simulation components as a cohesive model.

New technologies are currently being developed that could greatly reduce or even eliminate the need for users to specially configure their computers in order to run applications. This new technology will allow any user with an Internet-connected computer and a standard Web browser to load and run software applications, without considering the architecture or configuration of their computer. In addition, the software written for Web browser applications could be composed of objects or modules that don't physically exist on the user's computer, but rather on multiple locations across the Internet. This is the basis of the network-centric computing model.

When the concept of network-centric computing is applied specifically to simulation software, there are many benefits to the idea that a simulation model and the resulting output information could be available to virtually anyone in the world, instantly and on demand. Not only can the simulation analyst benefit from the output of a simulation run, but many others, perhaps with better knowledge of the current model input parameters, could initialize, run and view the simulation results. This may assist in

providing critical data for decision making. Additionally, with the concept of network-centric computing, a simulation model might be a collection of simulation objects that are physically located anywhere on the Internet, and are merely requested and delivered when the simulation model is executed.

The system model selected for this research is based on a semiconductor fabrication system. This environment was selected for several reasons. First, semiconductor fabrication is a complex environment that is difficult to model with many existing simulation environments and there appears to be a need for better simulation tools in this area. Second, the author is interested in developing an expertise in the area of semiconductor fabrication simulation modeling. Third, the close proximity of Oregon State University to Portland's "Silicon Forest" is an ideal opportunity to develop academia-industry research relationships.

The purpose of this research is to develop a methodology for running a network-centric simulation model over the Internet. The development of a prototype environment will provide the author with the ability to demonstrate the functionality of this concept and lay the groundwork for further development of a platform-independent, network-centric simulation environment.

1.2 Background of Study

1.2.1 Simulation

A simulation model is an abstract, logical, mathematical model of a real system which, when utilized, can create an artificial history of that system. Simulation is especially valuable when asking "what-if" questions about a system, where it is either too expensive or impossible to physically implement the changes necessary to answer the questions. Often, because of the repetitive and complex calculations, the use of a computer becomes invaluable in the execution of a simulation model. The main goal of simulation, then, is the development of a model that represents a real system such that experiments performed on the model yield results that would be, theoretically, exactly the same as performing the experiments on the actual system.

1.2.2 Review of Simulation Tools

There is a wide range of software used to exploit the computer as a tool in simulation modeling. The intention of this section is to give a brief history of simulation software, to introduce categories of software, and give a brief explanation of some of the most common software tools used in simulation modeling.

1.2.2.1 Evolution of simulation software.

In the early days of simulation, it was up to the simulation analyst to program every single calculation in a model. This was done in a general purpose programming language. This gave the simulationist great flexibility in the systems that could be modeled, but was very programming intensive. Often, the focus of the model was the coding of the program, and not the analysis of the system. However, with the development of more sophisticated simulation software, many of the standard functions are pre-programmed, leaving the simulation analyst to concentrate more on simulation, and less on program coding. The cost of this "ease of use" however, is a loss in flexibility. If a model can not be described in terms of the simulation software, then the simulationist has two choices. He/she can make some assumptions to transform the system into a form that could be modeled within the boundaries of the simulation software, or manipulate or "trick" the software into accurately representing the physical system. So, after several decades of specializing simulation software, recently the trend has been back towards the use of general purpose programming languages, using a recently developed programming paradigm called Object-Oriented Programming.

Since the start of its development in the 1950s, simulation software, just as all other computer technologies, has gone through many changes. Simulation programming that used to take an expert FORTRAN programmer, has now become a simulation application with a standard graphical user interface (GUI), that allows a simulation analyst with competent computer skills to generate large, complex simulations.

Simulation software can be categorized as one of the following:

- General Purpose Programming Language
- Special Purpose Simulation Language
- Simulators

1.2.2.2 General Purpose Programming Languages (GPPLs)

In the 1950's, when simulation began to develop into a technique for modeling systems, there was no software specifically designed for the simulation of systems. Therefore, simulation analysts used what was available, which was simply, programming languages.

There exists a dichotomy of the use of GPPLs for simulation. In support of using GPPLs, they are extremely flexible in the varied systems and detail that a proficient modeler can represent. Additionally, using a GPPL for simulation, it is likely that the language is used by many people, has the ability to execute on many different types of computers, and is probably already available on most computer installations. If one were to write a simulation program in FORTRAN, for instance, many other modelers would be able to read the program code and compile the code to run on their computer. So with general programming languages, programs have the potential of being highly distributable.

However, a disadvantage of using a general purpose programming language is the fact that the coding of the model becomes a very large task and requires the modeler to be

an accomplished programmer in the GPPL. Every aspect of the simulation model must be programmed. For instance, what if it was necessary within a model to generate a Poisson-distributed random variable? If a GPPL were being used to build the model, it would be necessary to create a subprogram that specifically generates a Poisson-distributed random variable. As will be discussed shortly, this is not the case with special purpose simulation languages.

Historically, some of the general purpose programming languages that have been used for simulation are:

- FORTRAN
- ALGOL
- PL/1
- C
- SIMULA
- LISP
- BASIC

In short any GPPL can be used as a coding environment to crate a simulation model.

1.2.2.3 Special Purpose Simulation Languages (SPSLs)

The next step in the evolution of simulation software, after the use of general purpose languages, was the creation of special purpose simulation languages. These

programming languages are designed specifically with the objective of simulating systems. They still require coding (programming) of the model, however many common simulation functions were added to the base language to speed model generation.

As an example of how SPSLs can help speed the creation of the model, consider the example from general purpose languages mentioned above. It was necessary in FORTRAN to write a subprogram to generate a Poisson random arrival. Special purpose languages come with this subroutine already integrated into the language. Therefore, when you want to write the code to generate a Poisson distributed random number, with a mean of 5 arrivals/hr., it may be as easy as:

Poisson(5)

By reusing the routines within the SPSL, obviously much time and effort can be saved and model generation becomes a less programming intensive task.

Although these languages made it easier and faster for an analyst to generate a model, the flexibility of modeling diverse and complex systems became restricted. With SPSLs, it was still possible to ask "what-if" questions about a system, as long as the questions were within the abilities of the SPSL.

Another problem with SPSLs was that simulationists still had to be competent programmers. According to Ball and Love (1994):

> Lengthy training is required to develop the skills required to both build and use a model based on a simulation language. A recent survey suggested that more emphasis was needed on the development of application methodologies and on improvements in education and training of potential users. An alternative view might be that existing simulation systems are too difficult to use and that efforts should be directed to improving usability. Here the term "ease of use" describes how well the simulation system matches the manufacturing or industrial

engineers' concept of the manufacturing system as well as the ease with which models can be created using the software.

The time required to build a model using a simulation language occupies a significant part of the total project duration. In particular, the stage of expression and conception of the logic is lengthy. The time taken is affected by the need to conceptualize the model logic (such as the sequence in which an operator runs a series of machines or the splitting of a batch across a number of machines) as well as the need to convert the logic into program code. However elegant the language, this approach remains constrained by the time required to develop the model logic. The model build time has a direct impact on the modeling system's utility in the design process. If the time required to evaluate a design using modeling equals or exceeds the time allocated to carry out the overall design task, then the model created can only be used to check the resulting design, not as an integral part of the design process. Hence the benefits of using simulation as part of the design process would be reduced.

Although the advantages of SPSLs were improved ease of use and faster model generation, there are some disadvantages associated with SPSLs. These languages are limited in the type and sophistication of real systems that they can be used to represent. Initially, these languages were very simplistic, and could model only the most basic systems. As they developed, however, they became more and more powerful. In terms of flexibility and their ability to model unique systems, special purpose languages may be less attractive than general purpose languages.

Many special purpose languages have been developed, yet only a few have gained widespread acceptance. Several of the most common languages are:

GPSS The General Purpose Simulation System (GPSS) was initially created early in the 1960's specifically to model discrete systems. This language was attractive in that it provides many common simulation functions internally, so the modeler can create models faster. A disadvantage was that GPSS was very limited in what it could model. It has evolved over the years to become much more flexible, however it still has modeling limitations.

GASP The General Activity Simulation Program (GASP) also started in the 1960's, and was further developed into GASP IV. GASP was a large collection of FORTRAN subroutines that were specifically designed for use in simulation models. GASP greatly eases the simulation of a system using FORTRAN. While GASP is now obsolete, it was an important step in the evolution of simulation software, as it led to the development of more advanced languages, specifically SLAM.

SIMSCRIPT Simscript was first developed by the RAND Corporation in the 1960's. It has evolved into the current version, Simscript II.5, and is still available commercially. It was unique in that it approaches a general purpose programming language, but includes very powerful simulation-specific functions. Simscript is a "natural language" simulation environment, where the simulation programs can be read in a sentence-like manner. Simscript is important because it took simulation languages to the next step of user-friendliness, and allowed a simulation analyst to concentrate less on programming, and more on the simulation study.

SLAM SLAM (Simulation Language for Alternative Modeling) is a high-level, FORTRAN-based special purpose simulation language. Its development started in the

1970s and its successor is still available commercially. Where it started as a language very similar to GPSS, it has continually developed and improved.

One distinct advantage of SLAM over other special purpose languages is the ability to model very complex or unique systems using FORTRAN. SLAM has the ability to pass information to and receive information from FORTRAN functions and subroutines, allowing the simulation analyst to call on the flexibility of a general purpose programming language when the requirements of the model demand it. While this adds flexibility to modeling, it also adds complexity and reduces ease of use.

One of the unique characteristics and a major advantage of SLAM is its use of a graphical interface to build models. SLAM provides a set of building blocks that allows an analyst to graphically represent a system as a "network." SLAM then translates the picture of the system into SLAM code that can be executed. This concept relieves the analyst even more of the task of coding a simulation model.

SIMAN SIMAN is a powerful, SPSL for modeling discrete, continuous and combined systems. Discrete-change systems can be modeled by using either a process-interaction or event scheduling orientation. Continuous-change systems are modeled with algebraic, difference, or differential equations. A combination of these orientations can be used to model combined discrete-continuous models. SIMAN is still commercially available and has evolved into a high level simulator called Arena.

1.2.2.4 Simulators

Simulators were the next generation of simulation software that once again, further removed the modeler from the task of programming simulation models. These are simulation applications that allow a modeler to build models often without writing a single line of programming code. Through the user interface, it is possible to construct models by selecting icons and using menus to enter data without having to be concerned with a programming language or conception of programming logic. (Ball and Love, 1994)

The development of simulators was an improvement and natural evolution of SPSLs. In fact, developing a GUI for an existing SPSL created most of the commercially available simulators. While these simulators may appear to overcome many of the problems and restrictions associated with SPSLs, the fact remains that the flexibility and amount of detail in a model is restricted by the finite definition of the SPSL. Additionally, Ball and Love point out a new problem. It seems with such an easy to use simulation model creation environment, aesthetically pleasing simulation models, with moving graphics, and real-time statistics output, just about anyone can create a simulation model, regardless of their system simulation expertise.

Several of the widely available commercial simulators are:

- Arena
- AutoMod
- ManSim

- ProModel
- SlamSystem
- Taylor II
- Witness

Even though simulators make it easy to create and run colorful, graphical models, the use of simulators still requires skill and ingenuity to represent the true behavior of a physical system. While simulators offer a number of advantages over other simulation tools (such as ease of use and less time to build models), they may be limited in the areas in which they have the potential to be applied. Limitations come as a result of limited-range functionally and difficulty in extension of the existing software.

1.2.3 The Return to General Purpose Programming Languages

Recently, simulation software research has come full circle, and currently, newer general purpose programming languages have been used to create simulation models. The difference this time, however, is the application of a new programming approach called Object-Oriented Programming, and new and more powerful languages to implement the OOP approach. The advantages of the new approach are the increased flexibility of using a general purpose programming language, manageability of the complexity of the system being modeled, and the ease of use of OOP languages when compared to the traditional general purpose programming languages.

Several of the commercially available and common OOP languages:

- C++
- SmallTalk
- Objective-C
- Object-Pascal
- CLOS

In the next section, a brief introduction to OOP is presented.

1.2.4 Object-Oriented Programming

"Object-oriented programming is a revolutionary idea, totally unlike anything that has come before in programming." (Budd, 1996)

Perhaps the most fundamental concept of object-oriented programming is best described by Floyd, (1989)

"Our world is filled with objects, so it seems only natural to describe and solve problems in terms of objects as well. This idea is the basis for object-oriented programming."

Although the object-oriented paradigm is a relatively new, popular concept in software engineering, the idea of programming based on objects was first developed in SIMULA, which is a simulation extension to the Algol-60 language. (Ozden, 1991).

Historically, most programmers have been educated mainly in the procedural paradigm. This would include the use of such languages as C, Pascal or FORTRAN. These languages share a common approach to solving a problem, differing only in their syntax.(Eldredge, 1990) Other less common programming paradigms would be the logic

programming paradigm, and the functional programming paradigm. (Budd, 1996) However, tools or design techniques have supported few of these paradigms.

A programming paradigm is literally defined as "an example or model" (Budd, 1996), but practically, provides the system designer with techniques that guide problem solution. Recently, there has been an interest in utilizing some of the alternatives to the procedural paradigm to facilitate the solution of certain types of problems. For example, the rule-based paradigm has been widely used in the development of expert systems. The access-oriented paradigm has proven to be a useful approach for building user interfaces. The object-oriented paradigm has also received intense attention recently as being useful in the pursuit of a variety of problem solutions. (Eldredge, 1990)

A major difference between the procedural paradigm and the OO paradigm is the focus of the programmer as the model is developed. In a procedural approach, the focus would be on an overall command loop which would be decomposed into subtasks as the model was developed. Data structures such as queues would be introduced as needed to support the overall algorithm. Using the OO paradigm, the main focus is on the description of objects and their behavior. Each object is defined abstractly in terms of a class. A class corresponds roughly to a type definition for a complex data type that includes fields for functions as well as fields for data. The actual objects in a problem solution are then represented as instances of these abstract classes. The instances are implemented as objects, which are independent regions of memory. In addition to identifying objects, the OOP paradigm also identifies relationships between these objects. These relationships help define the structure of the application design.

The OOP paradigm enables the decomposition of a software system into logical components that are represented as objects. Each object is a collection of data and procedures. The data represents the status of the object, and the procedures represent the behavior of the object. Theoretically, only the procedures owned by an object can access and change the value of its data. These features provide a convenient paradigm to develop models that closely resemble their real-world counterparts. An object-oriented program is executed through the exchange of messages that alter an object's status. The message-passing paradigm differs from a subroutine mechanism in that it allows a much greater degree of discretion concerning the interpretation of a message by the receiver. (Chang, 1994)

1.2.4.1 Components of Object-Oriented Programming

Since OOP is a relatively new area in software development, it is appropriate to present an introduction to the major concepts that make up the OOP paradigm. The concepts introduced in this section are the defining components of OOP, and they are common between all OOP languages.

Objects and Classes The principle idea of OOP is that everything is an object: variables, data structures, procedures, etc. An object is defined by the data areas it contains and the methods that manipulate its data areas. Objects that are created from the same definition are grouped together in a class. A class defines the data which is contained within the object, the manner in which the data is stored, the visibility of the

data to other objects, and the procedures which may perform operations on the data (Beaumariage, 1990). The actual object, conceptually in the computer's memory, is called an "instance" of class.

OOP languages contain five key concepts which result in making systems understandable, modifiable, and reusable (Beaumariage, 1990, Budd, 1996). These five concepts are: inheritance, encapsulation, message passing, binding, and polymorphism.

Inheritance is the ability to extend existing classes to create specialized structures. This is the basis of software reuse. All class definitions are part of a single hierarchical tree that defines the relationships between all objects in an class structure. At the base of the tree is the class definition of object, from which all subclass definitions inherit properties. From each of the subclasses, again, subclasses can be defined, inheriting properties from its superclass.

There are two important relationships between classes and subclasses. The IS-A relationship is the most important relationship in OO system design. It is the relationship that allows the programmer to use an existing class definition as the basis for a new definition. The IS-A relationship is a subclassing relation. This relation is implemented by an inheritance mechanism in OO languages. The inheritance mechanism is used to include the definition of a previously defined class(es) as the basis for a newly defined class that is a specialization of the existing class(es). An inheritance hierarchy results from successive uses of this specialization principle.(Eldrege, 1990) The inheritance hierarchy makes the definition of new classes more economical if they can be defined as specializations of existing classes. The new classes contain the existing code from the

older classes plus new specialized code. The deeper in the hierarchy, the more functionality is inherited by the new class.

A second structuring relationship in OO is the HAS-A relationship. Often as a new class is defined, its internal representation is defined in terms of instances of other existing classes. A class definition can be built by grouping non-related classes into a single class (Budd, 1996). The HAS-A relationship is also a very necessary component of OOP, as it allows great flexibility in object definition.

Encapsulation is the ability to combine data and the procedures that operate on that data into a single structure, then protect and hide this internals of this structure from other objects directly accessing and changing the data.. This means that an object's data and methods are enclosed within a tight boundary, one which cannot be broken by other objects.(Beaumariage, 1990) This protection of internal data structures is also known as information hiding (Budd, 1996). Therefore, an object's internal data areas, which are possibly very critical, cannot be modified directly by some external object, but only by the procedures explicitly defined by the object's methods. In other words, if you want an object to change its state, encapsulation prevents you from changing it yourself, but provides a mechanism to tell the object to change its own state. This ideal is a crucial concept in the verification of OO programs.

Message passing is the mechanism that makes encapsulation possible. It is the way in which all objects communicate with each other. Because an object's internal state may be very important, and it is not desired that any outside entity modify that state directly, the way to change the state is to pass a message to the object, in effect telling it

to change its state by way of one of its methods. Message passing is somewhat analogous to procedure calling in traditional programming languages (Beaumariage, 1990).

Binding refers to the process in which a procedure and the data on which it is to operate are related. Most procedural languages use early binding, in which binding is determined by the programmer and is performed when the code is written/complied. Declaring variables to be integer, real, logical, etc., is an example of the type of early binding done in traditional programming. Dynamic or late binding delays the binding process until the program is actually running. When an object receives a message, the OO language searches the object's class to find the method to perform. This use of late binding gives OOP a great deal of flexibility in several ways. First, it is possible for the data type of a variable to change during runtime (see polymorphism below). Another consideration is that different classes can have the same named methods with different code found in each object. Finally, many of the variables defined in the OOP environment can be independent of the data they actually contain at a particular instance during the runtime of the program (Beaumariage, 1990).

Polymorphism is the concept that a variable can hold different types of objects over the life of an executing program. By sending a single message to an object, the type of the object dictates how the message is interpreted and the request carried out. In other words, polymorphism is the ability to send the same message to different objects, with the result being that the object will respond in the intended manner (Budd, 1996).

These five concepts benefit OOP in several ways. First, understandability is achieved by grouping data and behavior into intuitive objects that represent real world

objects. An OOP object is the implementation of one complete concept and is, therefore, easier to understand. Second, modifiability is achieved because an object has all of the data and procedures associated with it tightly grouped together in one structure. When it becomes necessary to change the data or behaviors of an object, ideally, a programmer needs to only change the class definition of that specific object (Beaumariage, 1990). Third, reusability of code is achieved by the utilizing the IS-A relationship and the HAS-A relationship. With OOP, programmers no longer build entire programs from raw material, the bare statements and expressions of a programming language. Instead, they produce reusable software components by assembling components designed by other programmers. Brad Cox (1990) termed these reusable software objects "Software-IC's", as they resemble integrated silicon chips that can be "plugged" into many different circuit boards. They are an independent, fully operational component of the overall system. This is the true power of software development using the OOP paradigm.

1.2.5 Discrete-Event Simulation and Object Oriented Programming

Discrete-event simulation (DES) is a modeling methodology where state changes in the physical system are represented by a series of discrete events (Pritsker, 1986). A DES model assumes the system being simulated changes state only at discrete points in simulation time. The simulation model jumps from one state to another upon the occurrence of an event (Chang, 1994).

OOP is almost an exact match as a tool to develop DES models. Several obvious similarities exist between OOP and DES. Therefore, OOP has been used as a basis to develop many DES models. In support, Budd (1996) states:

> OOP allows a programmer to create a universe of well-behaved objects that courteously ask each other to carry out their various desires. This view of programming as creating a "universe" is in many ways similar to discrete-event simulation. In a discrete-event simulation, the user creates computer models of the various elements of the simulation, describes how they will interact with one another, and sets them moving. This is almost identical to the average object-oriented program, in which the user describes what the various entities within the universe of the program are, and how they will interact with one another, and finally sets them in motion. Thus, in object-oriented programming, we have the view that computation is simulation.

Further, Rosenberg (1992) states:

> Objects (in OOP) are a uniform programming element for computing and saving state. This makes them ideal for simulation problems where it is necessary to represent collections of things that interact.

There are special purpose simulation languages based on the object-oriented programming approach. For example, the SIMULA language is such a language and it was developed over twenty years ago. However, it has not gained widespread use for commercial simulations. This perhaps is due at least in part to the fact that it is an ALGOL based language and in many instances requires the writing of ALGOL subroutines in order to simulate a complete system.

One recently developed language based on the object-oriented approach was the SMALLTALK language (Goldberg, 1984). This language has evolved from artificial

intelligence applications. Although it may offer more promise than SIMULA of being widely used as a simulation language, it has not gained widespread use for commercial simulations.

The OOP language used for this research is Java, which is a new OOP language in its infancy. This is an exciting time, as Java has been said to have "taken the best of all OOP languages, and eliminated the worst" (Semich, 1996). The software development industry is in a frantic state to develop programs and new technologies that are related to the Java language. One purpose of this research is to explore the use of Java for implementing discrete-event simulations. Because Java is so new, very little research has been done in the area of discrete-event simulation using Java as the OOP language.

1.3 Literature Review

1.3.1 OOP Frameworks for Discrete-Event Simulation

A framework is a collection of OO class definitions that work closely with each other, and together, provide a reusable environment for a general category of problems. Many of the commercially available frameworks are designed for the creation of graphical user interfaces, or GUI applications (Budd, 1996). But the usefulness of frameworks is not limited to GUI design. Indeed, DES frameworks have been developed for a wide range of simulation applications.

One of the original and most extensive OO frameworks in the area of DES for manufacturing environments was BLOCS/M (Glassey and Adiga, 1990). Although the original research for BLOCS/M was started with a non-OOP language, C, the authors soon discovered that the OO approach was the path to follow, and the development language was switched to Objective-C, an OOP language. The objective of BLOCS/M was not to develop yet another general purpose simulation language for specific manufacturing systems, but rather to develop a framework for many diverse types of manufacturing. Coincidentally, the first systems they attempted to model were semiconductor fabrication systems, just as in this research.

There were two significant goals of Glassey and Adiga's work. First, to provide the necessary tools to make it easy to assemble simulation models customized for individual research questions. This tool would provide the ability to represent a system in a desired amount of detail. Often in research, the most difficult task is to be able to ask the correct questions. Sometimes, these questions are simple, but very abstract. It was thought that this framework could provide the means to ask these simple, but abstract questions.

The second goal was to make the classes in the framework easily modifiable and extensible. In this goal was the desire to make the framework a "starting point", where very detailed models could be developed by adding functionality to the existing classes by the use of inheritance.

The authors state that they achieved both of these goals. However, with regard to the first goal of making it easy to assemble simulation models, they state that it is

necessary to become proficient in the Objective-C programming language before a model can be built using their framework.

Another research project developed a similar framework for DES using the C++ OOP language (Eldredge et al. 1990). They found that the OO approach was very compatible with simulation methodologies, and allows for a simulation model to be written with a focus on the description of the problem rather than algorithms for solving the problem.

Yet another study (Beaumariage, 1990) developed a manufacturing simulation framework using the OOP language SMALLTALK. This was a very complete prototype environment that found the OOP approach to simulation modeling to be feasible and significantly beneficial.

However, the use of object-oriented programming is not likely to gain widespread use for simulation unless the language used is widely available and is familiar to a large number of simulation developers. Consequently, it appears that simulation through C++ is more likely to be successful in this regard than are other alternatives such as SIMULA and SMALLTALK. Each of these suffers from limited availability and a more restricted user base.

1.3.2 Simulation of Semiconductor Fabrication Systems

Semiconductor fabrication is a very complex manufacturing environment. (Hood, et al. 1989) After the blank silicon wafers have been manufactured, they undergo a

personalization process that transforms the wafer into usable integrated circuits. This process involves hundreds of processing steps at tens of tool groups performing intricate processes one or more times on the wafers in a predefined flow. Each workstation and individual tool has unique characteristics including process rates, batch sizes, process yields and reliability distributions. Each process step has unique characteristics such as tolerance in sub-micron units, critical defect density, uniformity, etc. Both product mix and customer demands drive setup and machine configuration to process different technologies on individual tools.

These characteristics result in a complex manufacturing environment with a high degree of variability. Variability causes many problems in manufacturing in general, including higher rework, lower yields, larger queues, reduced throughputs and longer turnaround times. This variability impacts the semiconductor fabrication function by hindering wafer movement, processing and storage.(Miller, 1994)

There are several characteristics that are common between many varied manufacturing environments, but are of particular importance in semiconductor fabrication systems. They are:

1. Raw Processing Time (RPT) or intrinsic processing time. This would be the time it would take a single wafer to be completely processed if it didn't have to wait at any of the tools. This is a constant dependent upon the specific product type and processing characteristics.
2. Product Turn Around Time (TAT) or cycle time. This is the actual time that elapses between when a wafer enters the system, until it has

completed processing. It will, of course, be longer than the RPT, as the wafer will be competing for finite resources within the system. One manufacturing objective would be to minimize TAT.

3. Throughput. This is the number of wafers per time unit coming off the line. A manufacturing objective would be to maximize throughput.
4. Work In Process (WIP) or Line Loading. This is the total number of wafers in the system at any given of time. A manufacturing objective would be to minimize WIP without affecting throughput.

While the definitions of these characteristics are straight forward, their interaction is very complex. The objective is to maximize the throughput of wafers and minimize the wafer's TAT while minimizing WIP. However, in an environment as variable and complex as a semiconductor fabrication system, operating procedures become very, very important to optimizing the system's operation. So, other than trial and error, how would successful operating procedures be developed?

Miller (1994) identifies several reasons why semiconductor fabrication systems are attractive candidates for simulation modeling. They are:

1. State-of-the art fabricators cost hundreds of millions of dollars to build, equip and staff. Fabricators must maximize the use of resources to achieve competitive unit costs and profitable levels of output.
2. Product TAT is a critical success factor in semiconductor manufacturing. TAT has a major impact on process control capabilities, yield rates, product contamination levels, and product costs. TAT is a key

measurement in virtually every semiconductor fabrication and must be managed successfully.

3. Given the importance of the relationship between throughput, TAT and WIP, it is necessary to develop operating procedures that will optimize these key characteristics simultaneously, while meeting other performance targets, including cost objectives, inventory levels, delivery commitments, process yields, and labor efficiency.

Since simulation provides an effective and powerful approach for capturing and analyzing complex manufacturing systems, semiconductor fabrication systems are an ideal candidate for simulation modeling. In fact, simulation may be the only practical technique available which is capable of evaluating different operating procedures for a specific semiconductor line (Miller, 1994).

1.3.3 Java and the Internet

Recently, the number of people accessing the Internet has grown exponentially. With this growth has come many new technological developments. First, of course, is the Web. Here, a Web browser, such as Netscape Navigator or Microsoft Internet Explorer is used to view documents posted on the Internet, written in the Hypertext Markup Language (HTML). These documents are more commonly known as "Web pages." The Web has proven to be an excellent media for the distribution of static textual and

graphical information to anyone in the world with a Web browser and a connection to the Internet.

More recently, in the drive to make the Internet more useful, newer technologies have been developed. One of those technologies is the use of the Java language to create programs that execute both across the Internet and inside the environment of a Web browser.

The hype surrounding the Java language gives the casual observer the impression that it is only suitable for writing small, graphical animation programs called "applets," that execute only within the environment of a Web browser. The explosion of such applets on the Web may support the notion that Java is not a serious language. In fact, Java is a powerful, well-designed OOP language that is an excellent platform for developing complex, object-oriented applications (Buss, 1996).

It is important to understand that Java was not developed specifically for use on the Internet and Web. It is a fully functional, stand alone programming language, developed by Sun Microsystems (Semich 1996). What gives Java the ability to interact with a Web browser is what is called the Java Virtual Machine (VM). The VM is a Java interpreter built into the code of a Web browser. Therefore, any browser that is said to be "Java-compatible" (i.e. Netscape Navigator, Microsoft Internet Explorer), has the capability to execute Java programs.

There are two revolutionary concepts that are associated with the Java VM. First, this makes Java a computer-platform independent language. A software developer has to code a program only once. Then, all computers that can run a Java-enabled Web browser

(virtually all computers) can execute the Java program, without the need to recompile code for the specific hardware, as is the case with many programming languages (Deitz, 1996). In other words, the same Java code can run on a PC, a Macintosh, a UNIX workstation or an IBM mainframe.

Second, the distribution and configuration of Java programs that run inside of a Web browser is greatly simplified. To run an applet, a connection to the Web server with the Java program code must be made, then, automatically and on the fly, the program is downloaded into the computer's local memory, where it is available for execution. When a new version of the software is distributed, the installation and configuration of the program is transparent to the user. This virtually eliminates the maintenance function of software installation and de-installation, and makes the software instantly available to anyone, anywhere with an Internet connection.

Because Java is such a new language, it needs maturing and the development of an infrastructure in the form of tools, frameworks, class libraries, and compilers. These are fast becoming available, and more robust and extensive support is on the way. One goal of this research will be to add functionality to the Java and simulation communities through the development of object classes and a simulation framework for use in simulating semiconductor fabrication systems.

1.3.4 Web-Based Simulation

Web-based simulation is so new, that it is not yet an existing field in simulation, but rather an idea which represents an interest on the part of simulationists to exploit Web technology (Fishwick, 1996). The research that has been published on Web-based simulation are prototype frameworks that allow analysts to create models, then post the models on a Web servers that deliver the Java-based applet to a requesting browser. Two research papers on this topic were published in 1996.

First, Miller, Nair, Zhang and Zhao (1996) developed "JSIM" at the University of Georgia. This is an OO software framework for creating Java simulation models using the process-based world view. They address the following seven key concepts in their work:

Simulation using graph theory. The underlying concept in the design of the JSIM library is graph theory. This is the idea that simulation models can be represented as a graph of nodes connected by a single edge. They state the benefits are the use of proven algorithms to test for deadlock and suitability of design for the model, and it provides an easy method for reuse when the models can be thought of as subgraphs.

Query Driven Simulation. This concept is based on the authors' previous work of storing simulation results in a database. This is done primarily because simulation results are generated at a large computational cost, and the results are worth saving. Then, when a user queries a simulation system based on QDS, the system first looks in the database for the results. If they exist, they are instantly reported. Otherwise, the model is

executed, and the results are returned. They state the benefits of this approach are reduced computational costs.

The Graphical Designer. The graphical designer is used to graphically build models. This removes the modeler from having to write any Java code. The graphical designer uses the Abstract Windows Tookit (AWT). (The AWT is the standard Java package used to create graphical user interfaces. It is included as part of the basic Java language.) While they state this is a key component of the framework, they don't discuss its implementation or use it in the paper's example. A graphical modeling building interface is definitely a beneficial aspect of a modeling framework. However, in JSIM, it seems that the GUI severely limits the flexibility of the environment, and may not be worth the effort applied to get such basic functionality. They list the graphical designer as part of their continuing efforts.

The JSIM structure. The JSIM class hierarchy is shown in Figure 1.1. They establish three Java packages. (A Java package is a method of organizing similar classes into a group.) First, the queue package is used for entity storage, the future event list, and simulation clock management. Second, the statistic package is used for the collection of statistics on the model. Third, the process package defines the possible functions that entities in the system can undergo. This is the method that implements the graph theory of simulation presented above.

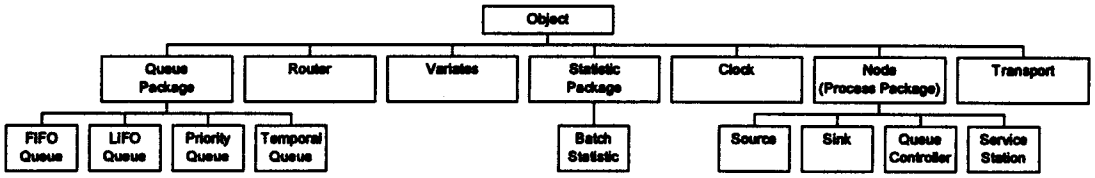

Figure 1.1: JSIM Class Hierarchy Diagram

Real-time and Virtual-time Simulations. One capability of the JSIM environment is the ability to run the model in a real-time mode, where graphical representations of entities move on the screen from node to node. They also describe the virtual-time mode, where the animation is turned off, and the simulation model is executed much faster. This allows the model builder to visually validate the model in real-time, then turn animation off for the lengthy, computation-intensive simulation runs.

Simulation of a Bank. As an example of the JSIM framework, the authors describe the necessary coding to build a simulation model of a bank. They don't use the graphical designer, but rather write the Java code using the JSIM framework.

Conclusions and Future Work. They believe that Web-based simulation will become an important technology in the future. The areas they list for future work are: the graphical designer, batching and unbatching of entities, and more complex routing of entities.

While JSIM helps the simulation and Java communities by providing another support tool, the following concepts, in the author's opinion, may not be widely applicable:

1. Simulation as an application of graph theory. While graph theory is an exceptional conceptual approach for some problems, (especially for computer scientists) it may limit the type of systems that can be modeled. For instance, in semiconductor fabrication systems, the routing of entities is a very complex function. Wafers have high process re-entrant behaviors, different routings dependent upon product type, and rework and defect characteristics. Graph theory may not be the correct approach for many simulation applications.
2. Simulation using the Query Driven Simulation (QDS) approach. No two systems are exactly identical, so no two simulation models of the respective systems should, theoretically, be identical. Therefore, it is not clear why the storing of simulation runs into a sophisticated database would aid the simulation analyst. This approach seems to add complexity to an area where considerable efforts to reduce complexity are constantly employed.

In addition to not embracing the above two concepts, this research will attempt to exploit the network-awareness of the Java language. Also, no graphical environment will be developed, as it may limit the functionality of the framework.

Another research effort in the area of simulation frameworks using Java is by Buss and Stork (1996). Their framework is called "Simkit," and is a very generalized environment that is intended to be extended for particular simulation applications. They state their goal was to provide simulation tools for the analyst and researcher that:

- are accessible to analysts without professional programming skills.
- are reliable enough for moderately sized projects.
- are capable enough for real problems.
- are conducive to rapid development of exploratory models.
- are low cost in money, programmer time and resources.
- promote code sharing and reuse.
- promote model sharing and reuse.
- allow for exploration of advanced simulation concepts, such as distributed simulation and remote entities.

This research implemented the OOP concept of software reuse to the extreme. Several public domain Java packages were used as part of the Simkit package structure. (See Figure 1.2)

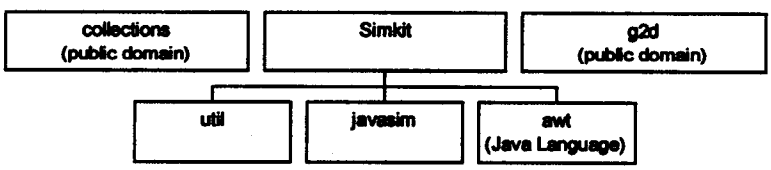

Figure 1.2: The Simkit Package Structure

Simkit consists of three Java packages. Utilities such as data structures, random variant generation, and statistics collection are implemented in the *util* package. Event-driven simulation is facilitated by the *javasim* package. Graphical User Interface

elements are implemented using the *awt* package, which is part of the basic Java language, available from Sun Microsystems. Additionally, two independently developed, public domain Java packages were utilized in the Simkit structure. The *collections* package is used for most data structures used in the Simkit framework. Charting capabilities are provided by the *g2d* (Graphics, 2-dimensional) package, and facilitate statistics display.

As an example of Simkit's implementation, an M/M/n queue model is presented in the paper.

The presentation of the Simkit environment is very limited and merely outlines the framework's package structure, without giving much detail on the implementation or class hierarchy. Therefore, it is difficult to evaluate the effectiveness of Simkit as a framework.

The concept of Web-based simulation presents at least two new benefits for creation, distribution and execution of simulation models (Buss, 1996):

1. Simulation models can be created and posted to a Web server so any user with a Java-enabled browser can execute the model. Since Navigator and Internet Explorer are the de facto standard for Web browsers, and are Java-enabled, the models will be widely available for use.
2. With the model instantly accessible anywhere in the world, a valid model may be of use to more users than in the past.

While the two benefits presented above are considerable, the purpose of this research is to explore the feasibility of developing a standard for network-centric simulation modeling using the Internet-based technologies introduced in this chapter.

1.3.5 Network-Centric Computing

The most common computer applications today are founded on the desktop-centric model of computing. In other words, the programs are developed with a specific computer architecture in mind (i.e. Windows on a PC, Macintosh or a UNIX-based workstation). In most cases, the software must be developed specifically for the intended platform. Often, "ports" from one architecture to another are cumbersome.

The Web and the Internet provide the basis for a technology that is changing the desktop-centric model. What is emerging is a network-centric model where platform independent standards like HTML and Java allow applications to be written once using these standards, and made available to any Web browser, without worrying about operating systems or platforms. At the same time, the users accessing these applications need not worry about the type of server supplying the applications or even what language the software is written in. This is possible, because they know that it must follow the standards, or be deemed useless by the rest of the Internet community (Flynn and Clarke 1996).

1.3.6 Network-Centric Simulation Modeling

Imagine you are tasked with upgrading your semiconductor manufacturing system. One of the decisions you are struggling with is the decision to buy a new quartz deposit machine or keep the one you currently own. The vendor of the new machine promises that the new machine will significantly improve the performance of your line, but how can you verify his claim?

What if you had developed a simulation model of your existing fab? This model was developed using a standard OO modeling framework, and each of the objects in the model representing the machines were developed not by your company, but rather by the machine vendors themselves. So you, as the simulation analyst, modify your model with the intention of testing the new machine's performance.

You edit your model and change the URL of the quartz deposit machine to point to the vendor's Web server, which contains the Java class that represents the quartz deposit machine (i.e. http://xyz.company.com/machines/quartzdep.class). You now run the model, utilizing the vendor-supplied simulation object, and the simulation results infer that, yes, the new machine does improve the performance of the fab. You can now make the decision to purchase the new machine with more confidence than merely the vendor's sales pitch.

This type of modeling could be possible for three reasons. First, because the simulation model was developed following an OO simulation standard. Second, the vendor has developed the machine simulation class according to the same standard. And

third, the model has the ability to pull its components from many different sources across the Internet. This is an example of what could be referred to as "network-centric simulation modeling."

1.3.7 Conclusion

This chapter has introduced and reviewed current and past research in several areas of simulation modeling. Specifically, six major concepts have been addressed:

1. Object-Oriented Programming
2. Object-Oriented Discrete-Event Simulation
3. Simulation Frameworks
4. Semiconductor Fabrication Simulation
5. Internet-Based Technologies
6. Network-Centric Simulation Modeling

All six of these concepts are applied in this research to develop a discrete-event framework for network-centric simulation modeling. In the next section, the goals, objectives and assumptions of this research are presented.

1.4 Research Goals, Objectives and Assumptions

The primary goals of this research are to:

1. Develop a prototype OOP framework for the implementation of network-centric simulation modeling, using a discrete-event simulation approach.

2. Propose a standard for OOP network-centric simulation modeling.
3. Utilize the framework to develop a network-centric simulation model of a typical semiconductor fabrication system.
4. Identify the direction of future research.

To achieve these goals, the following objectives are proposed:

1. Determine the simulation framework specification.
2. Determine the simulation model requirements. Because of the unique characteristics of semiconductor fabrication systems, the model structure and requirements must be defined. This includes determination of the model domain, modeling detail, and characteristics of the system.
3. Object class implementation. Once the general structures of the model requirements are determined, the definition and creation of the necessary classes can be undertaken. This allows objects such as machine types, transportation methods and statistics collection to be determined and implemented.
4. Development of a standard for network-centric simulation modeling. A detailed specification for the network-centric simulation modeling approach is created. This serves as a standard for simulation analysts to create network-centric simulation objects to be used in network-based simulations.
5. Model development. Once the components of the simulation model are in place, then they can be assembled into a functional simulation model.

This involves interfacing the simulation objects into the framework, with the result being a running model.

6. Identification of future research opportunities.

The principal assumption made in this research is that through the development of this single, specialized model and its innovative method of network-centric modeling, that the proposed concepts will lay the groundwork for a more generally applied simulation modeling methodology, and a robust standardization of the approach.

CHAPTER 2. RESEARCH PLAN AND PROCEDURES

To achieve the goals and objectives presented in the previous chapter, the research was performed in the four chronologically ordered phases: Environment Specification, Environment Implementation, Standard Development and Environment Application. The specific details of these four phases are presented in the next section.

2.1 Phase 1 - Environment Specification

Specification of a Network-Centric Object-Oriented Simulation Environment. In this phase, the conceptual and functional specification for a network-centric simulation environment was developed. First, the functionality needed to incorporate network-based simulation objects at runtime of the simulation model was identified. Second, the appropriate environment characteristics needed to accurately model a semiconductor fabrication system was documented. Finally, the interface for network-based simulation objects was identified. This leads to the development of the standard for network-centric simulation. The deliverable of this phase was a documented understanding of the functionality of the environment.

2.2 Phase 2 - Environment Implementation

Implementation of a Network-Centric Object-Oriented Simulation Environment. In this phase, the specification presented in the previous section was

implemented using the Java object-oriented programming language. At the completion of this phase, there was a functional software program capable of running a semiconductor fabrication system model using network-based simulation objects.

2.3 Phase 3 - Standard Development

Development of a Standard for Network-Centric Simulation. In this phase, the interface between the remote, network-based simulation objects and the simulation environment was documented. One of the design goals of the environment was to make the software easy to use. In order to have easy to use software, it is necessary to have complete and detailed documentation of the software. This documentation was created with a discrete-event, Java savvy analyst as the intended audience. However, no knowledge or prior experience with the Network-Centric Simulation Object System (NCSOS) simulation environment was assumed. The end result of this phase was a documented standard that allows an independent simulation analyst to implement an Internet-based simulation object.

2.4 Phase 4 - Application to a Target System

Application of the NCSOS Environment to a Target System. In this phase, the environment was used to create a simulation model of a target semiconductor fabrication system. This model was created by using a combination of both local and network-based simulation objects. Multiple network-based objects was created with the intent of

evaluating the difference in their behavior within the simulated system. The end result of this phase was a simulation model capable of executing simulation runs utilizing network-based simulation objects.

CHAPTER 3. ENVIRONMENT SPECIFICATION

Software development can be broadly categorized into two distinct functions; specification and implementation. This chapter presents the detailed specification for a network-centric, object-oriented, discrete-event simulation environment that is later used to model a semiconductor fabrication system. There are two unique concepts in this research that differ from most standard simulation environment research. First, this environment is intended to be "network-aware." This means that the simulation model does not consist of only locally stored software, but rather objects that may physically exist on a variety of computers all over the world. Second, this environment is not intended to be a "general-purpose" simulation environment, but rather a domain-specific environment, used to specifically model semiconductor fabrication systems. The reasoning behind this concept is that if a domain-specific environment can be successfully developed, then the knowledge gained in this research can be applied to later development of a general-purpose environment. This chapter presents the detailed specification of four specific areas in the following sections. Finally, the design goals of the simulation environment are identified.

3.1 Discrete-Event Simulation Environment Specification

Simulation Engine. The specification for a discrete-event simulation environment is relatively straightforward. If the state of a system under investigation

changes only upon the occurrence of one or more specific events, then the system can be said to follow the discrete-event paradigm. Therefore, a simulation model of such a system must have the ability to process and schedule events that change the system state. This is the basis for discrete-event simulation software.

The software developed in this research has at its center, a "discrete-event engine." Events that occur are scheduled in a Future Events List (FEL). The FEL chronologically orders the events according to their activation time. The FEL needs to be a robust structure that prevents logic errors such as out-of-order events, or the scheduling of events before the current simulation time. For example, it is not valid to schedule an event in the past.

The simulation is executed by processing these discrete-events, one by one, until there are no more events in the FEL, or the simulation run length is reached. Because each event is associated with an event time, the current simulation time is equal to the time stamp of each event. Of course, many events might occur at the same instance relative to the current simulation time, however, this is of no consequence. The state of the simulation model is changed at each event, and the current simulation time is increased according to the time stamp of each event.

The processing of each event is handled independently of the "simulation engine." The discrete-event software doesn't "care" what the event is, it just knows to trigger the event on the appropriate receiver at the appropriate simulated time.

Entity Representation. Traditionally, entities are created, flow through a simulated system, are queried for statistical properties, and are terminated or exit the system. This simulation environment is no different.

Physical Object Representation. Simulation software needs to have the functionality to represent physical objects such as machines, workcenters, transportation equipment, workers, etc. Since this environment generally focuses on manufacturing systems, the major physical representation is workstations. Entities flow from workstation to workstation. At each workstation, the entities consume resources and are manipulated by the processes at the workstation. This follows traditional simulation software design.

Statistics Collection. One of the main objectives of simulation modeling is to measure the performance of the simulated system. Therefore, the environment must provide functionality to collect and display statistics of interest. This simulation environment provides for the collection of several "types" of statistics.

First, the system must have the ability to collect and report entity-based statistics. These measures are specific to the individual entities, such as time in system, etc.

Second, the system must have the ability to collect and report system statistics. These measures are related to the overall performance of the system, instead of individual entities. These statistics collected for the system include measures such as work in process, cycle time, and throughput.

Since validation of a specific model is not the objective of this research, only a basic statistics collection functionality is created. Then, when further development of the

environment is undertaken, a more complete and robust statistic functionality will be created.

User Interface. The user interface for this environment is simple, yet complete. Basic functionality requires a way to start, stop, pause, resume, reset and quit the execution of a model. Additionally, a trace display must be created to help in the verification and validation of a model. Other GUI objects are created to provide for the network-related functionality in the form of entry fields, the ability to disable the trace, and the ability to set the delay between each event. The current simulation time and the simulation run length are displayed in the user interface.

3.2 Network-Centric Simulation Specification

The idea behind network-centric simulation is the concept that simulation models can be built by assembling independent simulation components into a cohesive model that can accurately represent a physical system.

In order to test this innovative modeling methodology, the simulation environment must have the functionality to import a software object that represents a physical workstation. The name and behavior of this workstation object is not be known until model runtime, when the name and URL of the workstation object are entered. The simulation environment will attempt to connect, over the Internet, to the workstation object definition. If this can successfully be accomplished, the model definition is

complete, and the simulation model can execute. If the location of the workstation object is invalid, or the object itself is invalid, the run terminates with an error.

This modeling methodology allows for a truly independent, object-oriented approach, where the software representation of a workstation is abstractly similar to the physical workstation. In other words, a simulation model is built in a similar fashion to the physical system; the objects are selected from a list of possible candidates, and logically connected to form a system.

3.3 Semiconductor Wafer Fabrication System Model Specification

The theoretical semiconductor fabrication system modeled in this research is graphically represented in Figure 3.1. It consists of seven workstations, each performing a different process to the wafer as it passes from workstation to workstation.

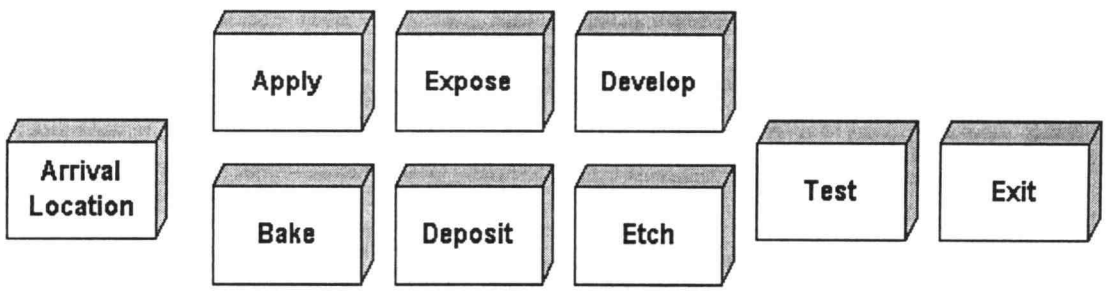

FIGURE 3.1: Target Semiconductor Fabrication System

The main characteristic that semiconductor fabrication systems display that is different from many other manufacturing systems is the complex routing that entities follow as they flow through the system. The number of sequence steps can easily reach into the hundreds, and the sequence itself can vary greatly from one wafer type to the next. Typically, there is a high product mix of wafers that are processed in a particular system. While the physical wafers appear to be of similar dimension, their routing through the system can vary greatly. It is this complex, re-entrant route through the system that has traditionally been difficult to model.

This simulation environment attempts to solve this problem by including routing information into the entity itself. This appears to be an intuitive location to store such entity-specific information. Each entity is of a certain type, and it has the ability to keep track of its state in the processing sequence. When a step in the process has been completed, the environment merely "asks" the entity which workstation is next in its processing sequence (Beaumariage, 1990). This eliminates the need to create and maintain separate routing and status information about each entity as it processes through the system.

Additionally, entities store and update entity-specific statistics. Again, this makes intuitive sense, and eliminates the need to create additional structures to store, track and report statistics for each entity.

3.4 Network-Centric Simulation Standard Specification

If the functionality of the network-centric simulation environment is to be useful to anyone other than the author, there must be documentation, or a standard, which allows other simulation analysts to independently create objects that represent workstations and function within the environment.

Specifically, this standard describes the interface "rules" that must be followed if an independently created object is going to interact with the base simulation environment. It describes the basic, default behavior of a workstation, the legal communication protocol between the workstation and the simulation engine, and such details as how to report statistics. Additionally, there are some strict limitations of the functionality of a workstation object during this initial study of the feasibility of network-centric simulation. These limitations are included in the documentation of the standard.

3.5 Design Goals

The following five design goals were pursued throughout the implementation of the specifications discussed above.

1. Ease of use. An intentional effort was extended to make the entire system as easy to use as possible. A simulation environment is of little use if the author is the only person with the understanding of the software to use it in practice.

2. Create simple objects. Each object definition in the framework is as simple as possible and implements only a small number of significant behaviors. There was an effort to use more simple objects instead of fewer, more complex objects.
3. Incorporate direct abstraction into objects. Object-oriented programming has excellent data abstraction capabilities. Whenever possible, abstraction into the simulation objects that represent physical objects was emphasized.
4. Create a framework that is easy to modify and extend. Object-oriented programming promotes these concepts. Whenever possible, code was created that is extendable and easily modified.
5. Create independent objects. Because this is a network-centric environment, it was of prime importance that the software objects that represent physical objects be developed with independence in mind. These remote objects will be created, tested and stored in a remote, isolated location, so their design should allow for data and behavioral encapsulation.

CHAPTER 4. ENVIRONMENT IMPLEMENTATION

The second distinct function of software development, after specification, is the implementation of the specification. In this chapter, details about the implementation of the environment are presented. First, an overall description of the structures and objects that were used to implement the environment are presented. This describes the general object-oriented structure and major components of the environment. Second, the implementation of the semiconductor fabrication model is presented. This section emphasizes the use of the environment's classes to model the specific system that was previously described.

Java was the programming language used for implementation of this environment. It was selected for its ease of use and its network programming capabilities. Java is quickly gaining worldwide popularity and new Java technologies and functionalities surface almost weekly. The Java-specific syntax is not emphasized in this chapter, but can be located in Appendix A.

In this section, the specific names of classes that have been implemented into the environment have a bold and italic typeface. For example, ***SimEvent*** is the actual name of a class that was implemented in the environment. Also, each class definition in the environment has an associated CRC (Component, Responsibility, Collaborators) card. The CRC card is common practice among software developers, as it graphically represents and documents the definition, behavior, and communication protocols of each software component (Budd, 1996).

4.1 General Structure

4.1.1 The Environment Class Structure

The environment execution class diagram is shown in Figure 4.1. These are the significant classes that make up the NCSOS environment. In the following sections, this structure is broken down into the logical sub-structures that present a description of the classes in detail.

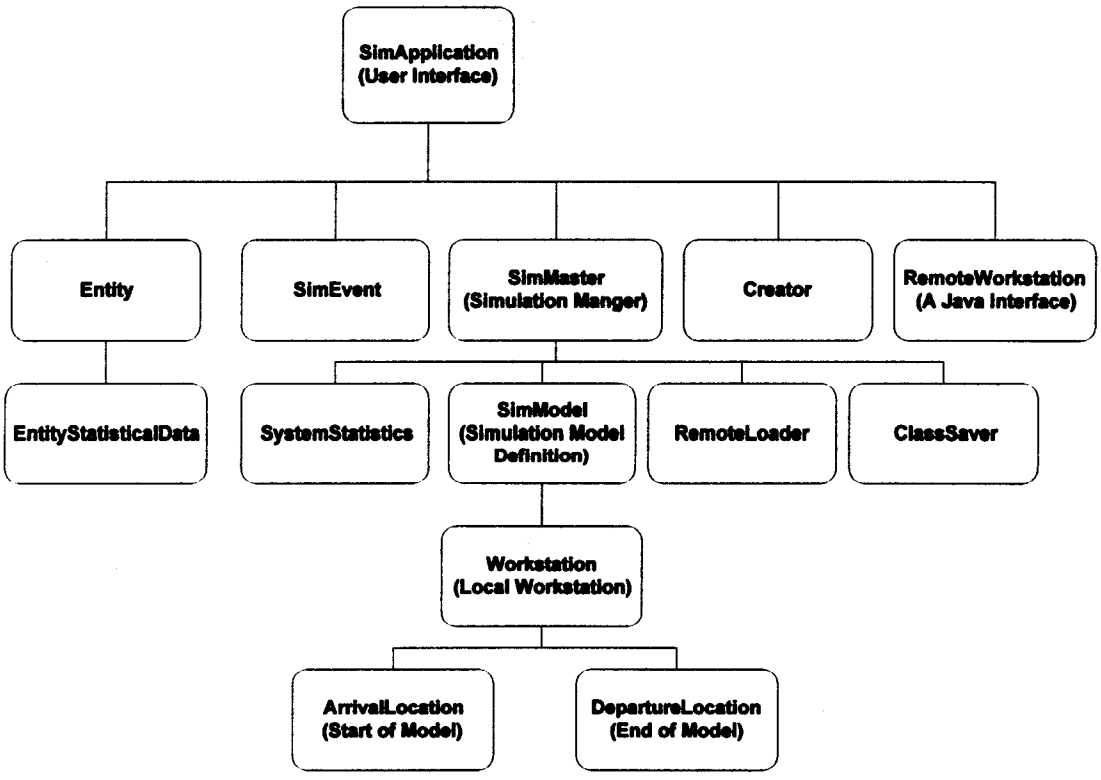

Figure 4.1: The NCSOS Execution Diagram

4.1.2 The Discrete-Event Simulation Substructure

The discrete-event simulation class hierarchy of the environment is given in Figure 4.2. This substructure implements the discrete-event simulation functionality of the entire environment. Each class definition is a subclass of the Java ***Object*** class. (All classes in Java are a subclass of ***Object***.) ***SimMaster*** is the manager of the simulation, and coordinates and controls the execution of the discrete events. ***SimEvent*** represents the discrete events themselves. ***Entity*** implements the concept of physical parts moving through the system by visiting locations such as workstations, inspection areas, etc. ***EntityStatisticalData*** is a logical collection of statistical data and behaviors that is related to a specific entity. Finally, ***SystemStatistics*** is the object that maintains and manages statistics related to the entire system.

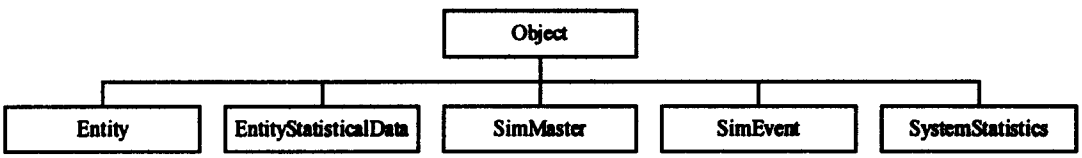

Figure 4.2: The Discrete-Event Simulation Class Hierarchy

The ***SimMaster*** class is at the heart of the simulation engine. It is responsible for management of the entire simulation run. It manages the FEL by scheduling and processing events, keeps the current simulation time, and coordinates most of the

communication between objects that are not directly linked. Its CRC card is presented in Figure 4.3.

SimMaster - Manager of Simulation	
Responsibilities	**Collaborators**
•Initialize the SimModel •Start the simulation •Keep simulation time •Create/Initialize SystemStatistics •Schedule SimEvents into the FEL •Remove SimEvents from the FEL and send the SimEvents to the workstations •Coordinate communication between the SimModel and the User Interface	•SimApplication •SimModel •Workstations •SimEvents •Entities

Figure 4.3: The ***SimMaster*** CRC Card.

SimEvent is the environment's representation of an event that changes the state of the simulated system. Each instance of ***SimEvent*** has four instance variables that fully define the event's intended purpose. Each ***SimEvent*** has a time stamp, an entity, a receiver and a name. The time stamp is the simulation time that the event will occur. The entity variable holds the instance of ***Entity*** that is related to this event. (The entity is an optional variable for the ***SimEvent***, as some events are not directly related to an instance of ***Entity***.) The receiver in this environment is the ***Workstation*** that will actually

process the event. Finally, the name variable holds the type of event that will be processed at the receiver. This is a dynamically, late bound variable that is only determined during the execution of the model. The CRC card for SimEvent is shown in Figure 4.4.

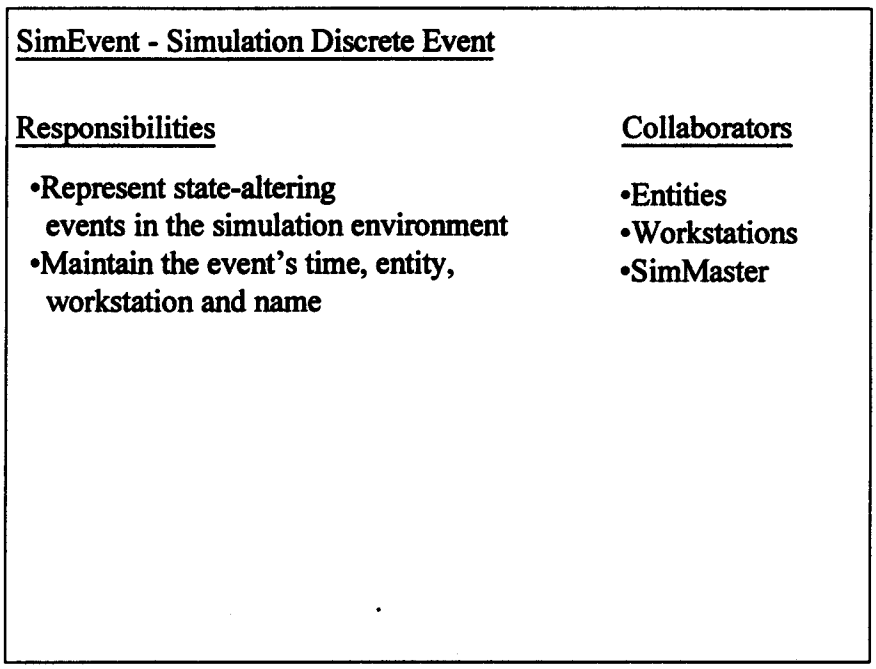

Figure 4.4: The ***SimEvent*** CRC Card.

Entity is the environment's representation of objects that flow through the system. Because of the characteristics of the intended use of the environment, the entities are moderately complex objects, with many variables. Each entity has a unique identification number, a representation of its route through the system, its processing time at each step, and contains an object that holds statistical data (see ***EntityStatisticalData*** class). The

entity also keeps track of its current step in the routing sequence. The ***Entity*** CRC card is given in Figure 4.5.

Entity - Simulation Entity	
Responsibilities	**Collaborators**
•Maintain entity state	•SimMaster
•Maintain entity id	•SimEvent
•Maintain entity routing	•Workstation
•Maintain entity processing times	•EntityStatistical-Data
•Create EntitiyStatisticalData object	

Figure 4.5: The ***Entity*** CRC Card.

The ***EntityStatisticalData*** class contains all of the statistical data and methods to efficiently manage an entity's statistics. This data and behavior could have easily been included with the ***Entity*** class. However, good object-oriented design encourages simple and concise object classes. This led to the separation of statistical data from the entity itself. The CRC card for the ***EntityStatisticalData*** is shown in Figure 4.6.

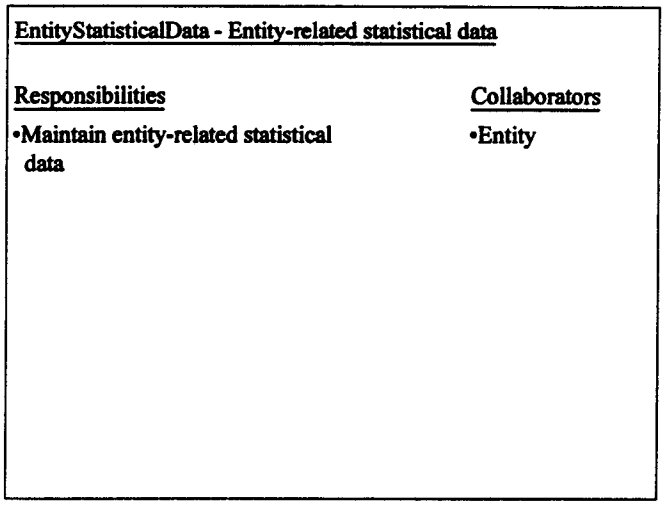

Figure 4.6: The ***EntityStatisticalData*** CRC Card.

SystemStatistics is the class that collects, maintains and reports the measures of performance that are defined. It is intended to be the manager of statistical information for the system. Its CRC card is shown in Figure 4.7.

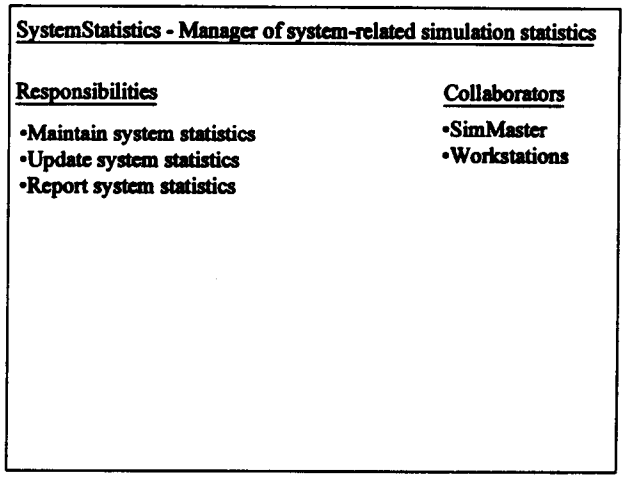

Figure 4.7: The ***SystemStatistics*** CRC Card.

4.1.3 The Simulation Model Substructure

The Simulation Model substructure is shown in Figure 4.8. The ***SimModel*** is the class where the layout of the model is defined. This layout would include instances of locally defined workstations that are known before the model is compiled. The ***Workstation*** class defines the default behavior common to all workstations. The ***ArrivalLocation*** and ***DepartureLocation*** are essentially the entity "source" and "sink" respectively. It can be seen that these are specialized workstations, and therefore are subclasses of the ***Workstation*** class. The ***RemoteWorkstation*** interface provides the functionality to dynamically create and include workstations into the simulation model that are not known at compile time, but are entered at runtime of the model. Each of these classes and CRC cards are presented in more detail below.

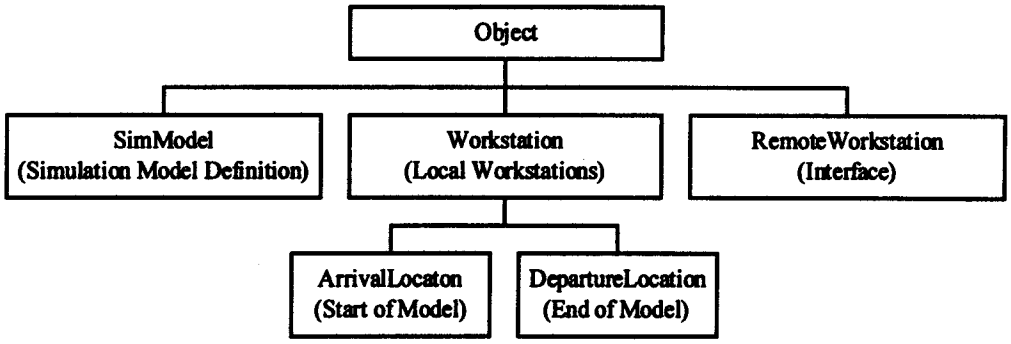

Figure 4.8: The Simulation Model Substructure Class Hierarchy

The ***SimModel*** class defines the configuration of the simulation model layout. This is where the workstations would be incorporated into the simulation model by defining instance variables. The modeler would also create place holder variables to allow a ***RemoteWorkstation*** to be created at model runtime. During execution of the model, the network substructure would be used to locate and create the ***RemoteWorkstation*** objects. The CRC card for the ***SimModel*** can be seen in Figure 4.9.

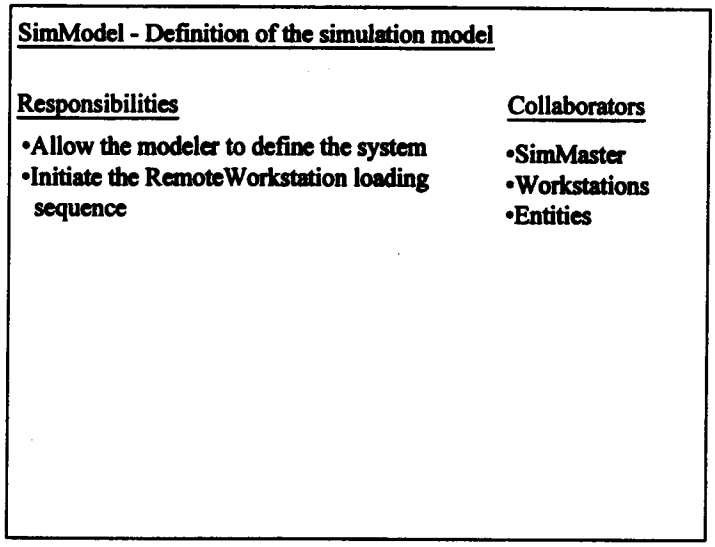

Figure 4.9: The ***SimModel*** CRC Card.

The ***Workstation*** class is a very critical component in the environment. It defines the default behavior for all workstations. Whenever a new type of workstation is defined, it will be a subclass of the parent ***Workstation*** class. This is true for both locally and remotely defined workstation objects. Much of the communication that each specialized workstation has with the environment is implemented through the ***Workstation*** class. For instance, the parent ***Workstation*** class always handles all of the scheduling of ***SimEvents*** by individual workstations. The CRC card for the ***Workstation*** class is presented in Figure 4.10.

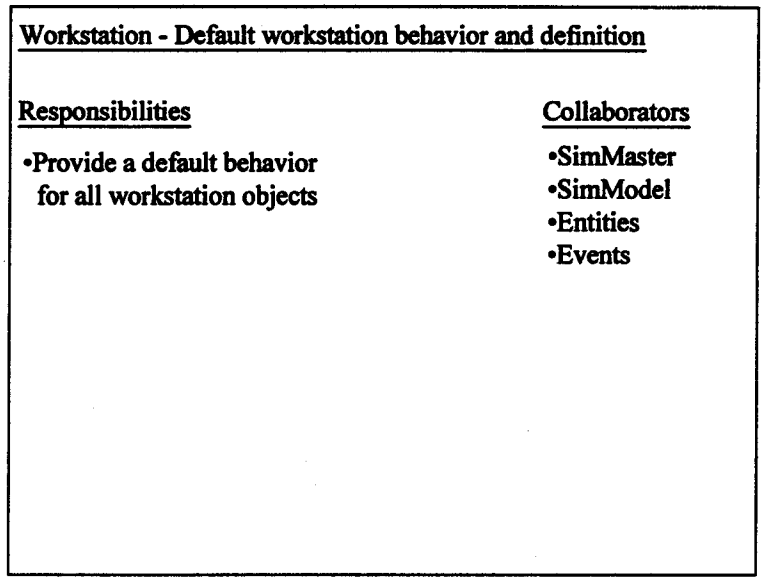

Figure 4.10. The ***Workstation*** CRC Card.

The ***RemoteWorkstation*** is not a class definition, but an interface. In Java, an interface allows for an object to promise that it will implement certain behaviors, without exposing exactly how this behavior will be implemented. This functionality allows for a remote object to be dynamically imported into the model at runtime. Therefore, each remote workstation object must implement the ***RemoteWorkstation*** interface. The CRC card for the ***RemoteWorkstation*** interface is shown in Figure 4.11.

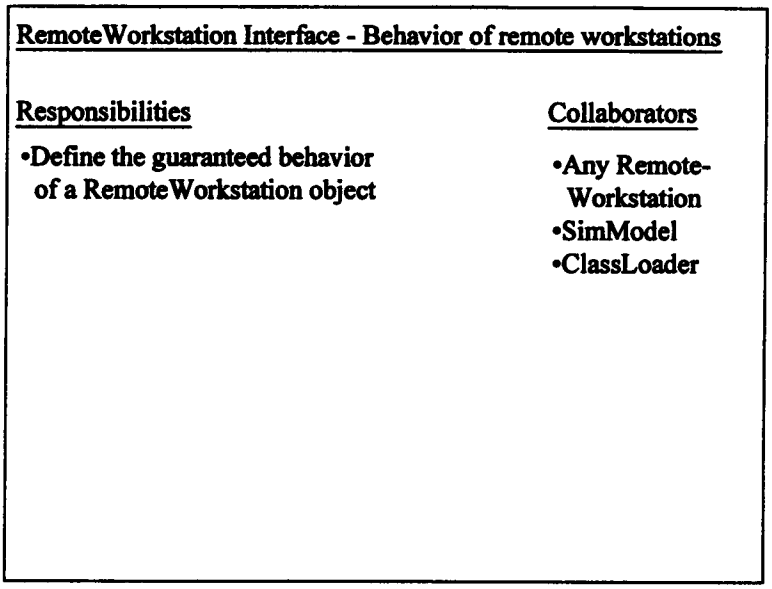

Figure 4.11: The ***RemoteWorkstation Interface*** CRC Card.

4.1.4 The Network Substructure

The network substructure is responsible for the creation and initialization of the remote workstation object at runtime of the model. These events only take place at the very beginning of a simulation run. In essence, the simulation model is incomplete until the network substructure can find the remote object on the Internet, download its definition, and create an instance of the object. ***ClassSaver*** is the class that is responsible for locating and downloading the ***RemoteWorkstation*** class definition. Then, the ***RemoteLoader*** will create an instance of this object, and insert it into the simulation model. At this point, the simulation model is complete, and execution of the simulation run can begin. The network substructure can be seen in Figure 4.12.

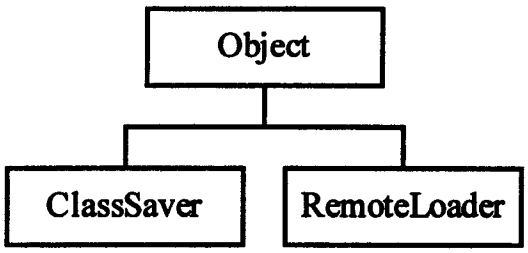

Figure 4.12: The Network Substructure Class Hierarchy.

The ***ClassSaver*** has the responsibility of locating and downloading the remote workstation definition from the Internet. This is a relatively simple procedure, and causes the simulation run to abort if the user-defined object can not be located and downloaded. The CRC card for the ***ClassSaver*** is shown in Figure 4.13.

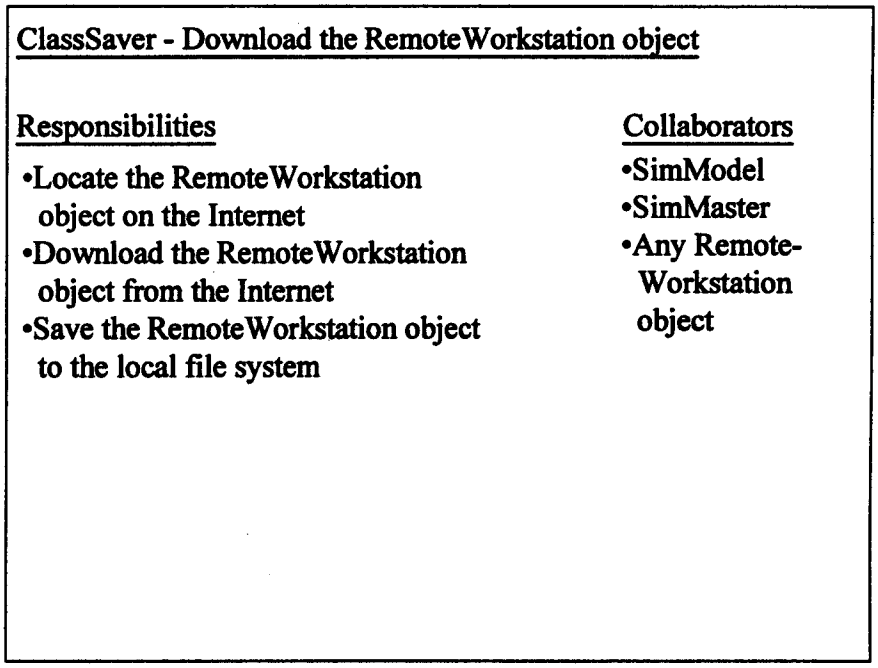

Figure 4.13: The ***ClassSaver*** CRC Card.

The ***RemoteClassLoader*** has the responsibility of creating and initializing an instance of a remote workstation. The CRC card for the ***RemoteClassLoader*** is seen in Figure 4.14.

RemoteClassLoader - Creates an instance of a RemoteWorkstation	
Responsibilities	Collaborators
•Creates and initializes an instance of a RemoteWorkstation	•SimModel •Any Remote-Workstation

Figure 4.14: The ***RemoteClassLoader*** CRC Card.

4.1.5 The User Interface Substructure

The ***SimApplication*** class is the sole user interface definition. It defines how the user interface appears, and how it responds to input from the user. It also handles all of the output from the simulation model to the screen. Most of its interaction with the simulation environment is accomplished through the ***SimMaster*** class. The CRC card for the ***SimApplication*** is found in Figure 4.15.

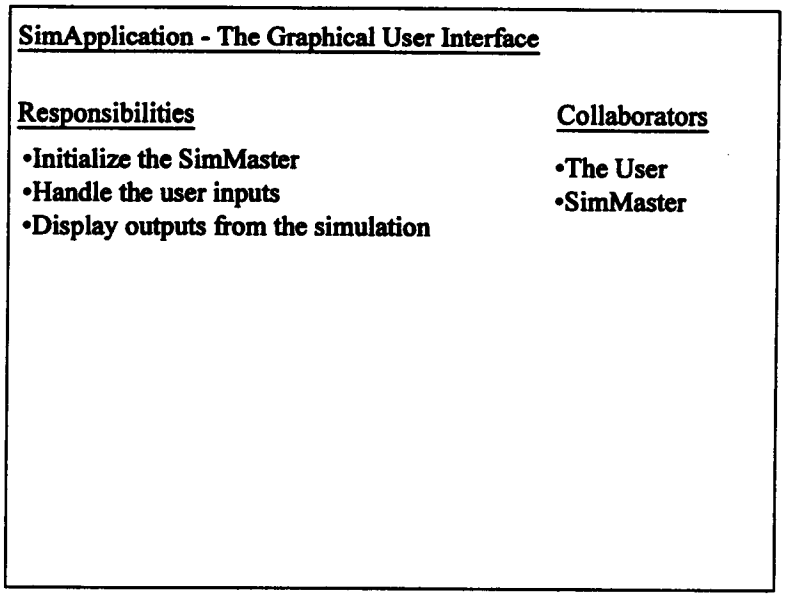

Figure 4.15: The ***SimApplication*** CRC Card.

4.2 Model Implementation

The Model implementation is intentionally very straightforward. There are five phases to the implementation of a semiconductor fabrication system model within this simulation environment.

4.2.1 Phase 1 - Define Behavior of All Local Workstations

A graphical representation of this system can is found in Figure 3.1. The first phase in the implementation of the semiconductor fabrication system model was to identify and define the structure and behavior of each specific workstation, except for the

"Expose" workstation (see section 4.2.4 below). A new class definition was created for each of the following six: Apply, Develop, Bake, Deposit, Etch and Test. The Java class definition files are located in Appendix B.

In each of these class definitions, the class Workstation was subclassed to get the common behavior of all workstations in the environment. Then, the powerful abstraction capabilities of object-oriented programming were used to define the unique behavior of the workstations.

4.2.2 Phase 2 - Define Model Layout

In the second phase, the layout of the model was defined in the ***SimModel*** class definition file. An instance variable was created for each of the six specific workstation objects defined in phase 1. Also, a variable type of ***RemoteWorkstation*** was created to hold the instance of Expose workstation that will not be known until it is imported into the model at runtime. This ability to create a variable with an unknown type is one of the innovative characteristics of this environment, and a very powerful tool in object-oriented programming. When the model is initiated at runtime, an instance of each of the local workstations will be created and loaded into their corresponding instance variable in the ***SimModel*** object. Additionally, the network substructure of the environment will locate the Expose ***RemoteWorkstation*** on the Internet, create an instance and load it into the ***SimModel's*** instance variable of the ***RemoteWorkstation*** type. This will complete the model, and it will be ready to execute the run.

4.2.3 Phase 3 - Define Behavior of Entities

The third phase of model implementation was the definition of the structure and behavior of the entities that represent wafers passing through a fabrication system. The class definition file of the Entity class can be found in Appendix A. As defined in the model specification in chapter 3, the wafer entities needed to define and maintain a significant amount of data and behavior. Therefore, the entity was defined with a sequence of workstations it must visit, with the corresponding processing time at each of these sequence steps. The entity had to also maintain its current stage in the processing sequence, so that when processing was complete at a specific workstation, the entity could route itself to the next workstation in its processing sequence.

Additionally, statistics collection of entity-related data was incorporated into the class ***EntityStatisticalData***, which was defined to collect appropriate measures of performance for wafers in this system.

After phase 3 is complete, the model is completely defined and ready to execute, except for the behavior of the unknown Expose workstation. This final component of the model will not be known until the simulation application is started, and the remote workstation object is identified in the user interface.

4.2.4 Phase 4 - Define Remote Expose Workstation Behavior

The Expose workstation was not implemented as a local workstation, but rather an Internet-based ***RemoteWorkstation***. Because this function was intended to be implemented by an independent simulation analyst, a specific document was created for

just this purpose. Because of the document's length, it has been included as a separate chapter (see chapter 5).

4.2.5 Phase 5 - Identify URL of Remote Workstation

The final phase of model implementation could not be completed until runtime of the model. Prior to this phase, an independent analyst must have implemented the creation of a remote workstation object. With the assumption that a valid remote workstation object existed, the URL and name of the remote workstation object was entered into the user interface. At this point, the model was now completely defined, including an instance of the previously unknown remote workstation object. The simulation model was now ready to run.

CHAPTER 5. NCSOS STANDARD

5.1 Introduction

This chapter includes the documented standard for network-centric simulation using the NCSOS environment. The objective of the document is to allow simulation analysts to independently create remote workstation objects in an isolated environment. The premise of the document is that if the analyst will adhere to the standard, then remote workstation objects will seamlessly interact with the Simulation Module, allowing a model to execute. The document was originally created in HTML, and posted on a Web server, allowing for easy and rapid distribution to simulation analysts. The following section presents the standard, formatted for use in this context.

5.2 Network-Centric Simulation using the NCSOS Environment

This document introduces the NCSOS simulation environment, and explains how to independently create objects that will interact with the Simulation Module when a model is executed.

5.2.1 Introduction to the NCSOS Environment

The NCSOS is a prototype discrete-event simulation environment that will allow a simulation modeler to build simulation models using a combination of local and remote

Internet-aware software. It was written in the Java object-oriented programming language, using Sun Microsystems's JDK, version 1.1.

The "Simulation Module" (see Figure 5.1) is used to run a simulation model. The modeler will fully define a simulation model, except for one or more workstations. Then, at runtime of the model, the names and URL's of the missing workstation objects will be entered. The workstation objects will then be loaded over the Internet and will interact with the simulation environment as if the workstation objects were part of the original environment definition. One design emphasis for this environment was the standardization of the interface between the remote object and the Simulation Module.

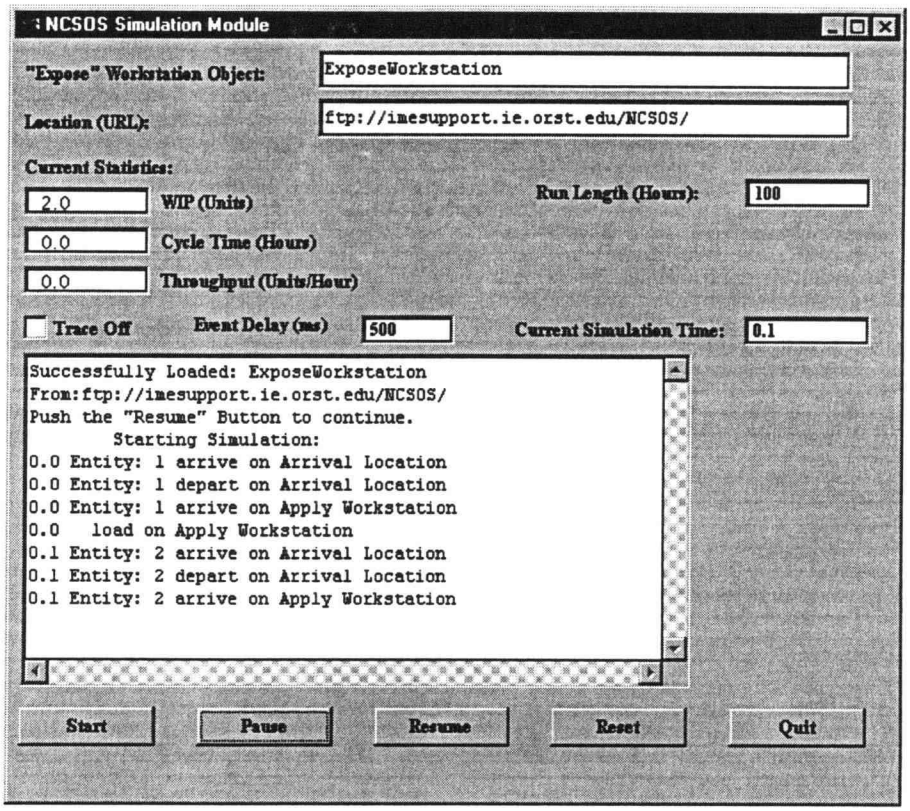

Figure 5.1: Simulation Module User Interface

A simulation analyst (different from the modeler) creates and tests simulation objects in an isolated environment. The analyst will define the name and behavior of a workstation simulation object according to the characteristics of the workstation that is being modeled. Then, this object can be posted on the Internet, and interact with the Simulation Module during a simulation run.

This document is intended for the simulation analyst that will independently create a remote workstation object who will interact with the "Simulation Module." It describes how the environment is structured, and the interface to the simulation

environment. In order to use this document, the independent simulation analyst will need to:

1. Read the rest of this document.
2. Download and install the JDK, version 1.1 (or later) from Sun Microsystems.
3. Download and install the NCSOS software.

5.2.2 Potential Usefulness

In a fully developed environment, an analyst could build a simulation model that consists of a base simulation framework and a collection of network-aware simulation objects that resemble such physical objects as workstations, entities, transportation equipment, warehouses and inspection stations. The modeler would be able to "assemble" the model by selecting a combination of pre-defined simulation objects that each simulate the behavior of the actual equipment they represent.

One advantage to this modeling approach is the ability to build a simulation model using one or more simulation objects developed by independent vendors, which simulate the performance and behavior of their company's equipment. For example, consider the following simplified semiconductor fabrication system (Figure 5.2).

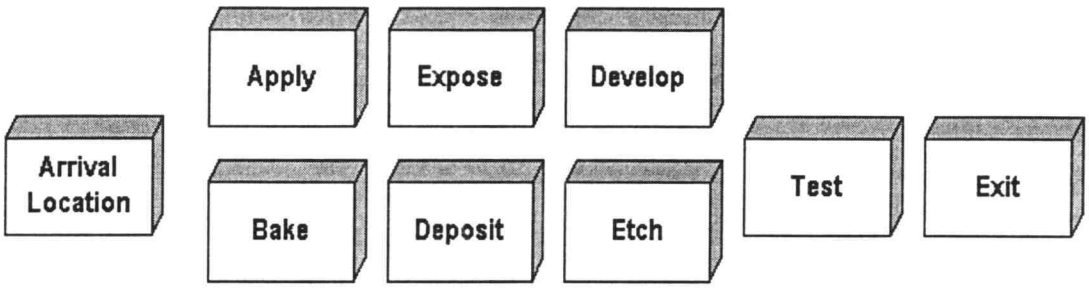

Figure 5.2: Semiconductor Fabrication System.

Suppose in this system, that both vendor A and vendor B offer an "Expose" workstation. Also imagine that both of these vendors have developed a simulation object, using the NCSOS, that resembles the behavior and characteristics of their respective machines. A system analyst may want to decide which vendor's machine will work best in a production system.

First, a set of simulation runs is performed using vendor A's object and the appropriate measures of performance are recorded. Next a set of simulations runs is performed replacing vendor A's simulation object with vendor B's object. This scenario should give some insight on which machine would be more beneficial to the system.

5.2.3 Prototype Environment

The objective of this prototype environment was to determine if the methodology of network-centric simulation was feasible. In order to achieve this objective, a discrete-

event simulation environment was built using the object-oriented programming language, Java.

Part of the functionality of this framework is the ability to dynamically load, at runtime, software objects from resources located at a URL on the Internet

In this simulation environment, a simplified model of a semiconductor fabrication system was built. The model was defined, except for the name and behavior of the "Expose" workstation. The simulation model will not run until an instance of a valid software object representing an Expose workstation can be created.

5.2.4 Objective

The objective of using this standard is to "build" a simulation object that simulates the behavior of an "Expose" workstation. Once this class definition has been compiled and tested by the analyst, the object will be posted on an ftp server. The independent analyst will send the author the URL of the workstation object. Then, the NCSOS Simulation Module will be started, the URL and name of the "Expose" workstation object will be entered, and the model will attempt to execute.

This document includes a description of the interface that the "Expose" workstation must use to communicate with the Simulation Module. By following the "rules" of the interface, interaction of a remote "Expose" workstation object with the Simulation Module is possible.

5.2.5 Qualifications of Simulation Analyst

There has been a specific attempt to keep the requirements for an independent analyst to a minimum. In order to develop a workstation that will interact with the Simulation Module, the analyst must have:

1. A basic understanding of how discrete-event simulation software functions. This includes such concepts as the event calendar, scheduling events on the event calendar, and processing events as the simulated time is advanced. It is not necessary to know "how" to perform these functions, just an understanding of the concepts.
2. An intermediate level of competency using the Java language. This would include basic OOP programming concepts such as: inheritance, message passing syntax, and the ability to maintain the state of a software object using instance variables.

5.2.6 More About the Simulation Module

The Simulation Module is the part of the environment that has the ability to dynamically load objects over the Internet and run simulation models. The independent analyst will download a simplified version of the Simulation Module to help in the development and testing of the "Expose" workstation object.

Events that change the system's state are the driving force in discrete-event simulation. In the NCSOS environment, there is a class called "SimEvent" that represents events in the simulation. Each event consists of:

1. The **time** of the event.
2. Optionally, the **entity** associated with the event.
3. The **workstation** on which the event occurs.
4. The **name** of the event that occurs.

These events are stored in a future events list, that sort the events in chronological order, and process the events one by one, as the simulation time is advanced. The simulation manager or "SimMaster" manages the future events list by scheduling events into the list, and sending the events to the appropriate objects when the appropriate simulation time occurs.

The SimMaster also manages all of the other simulation- related activities, delegating tasks to the appropriate objects. A simplified relationship between the major simulation objects can be seen in Figure 5.3.

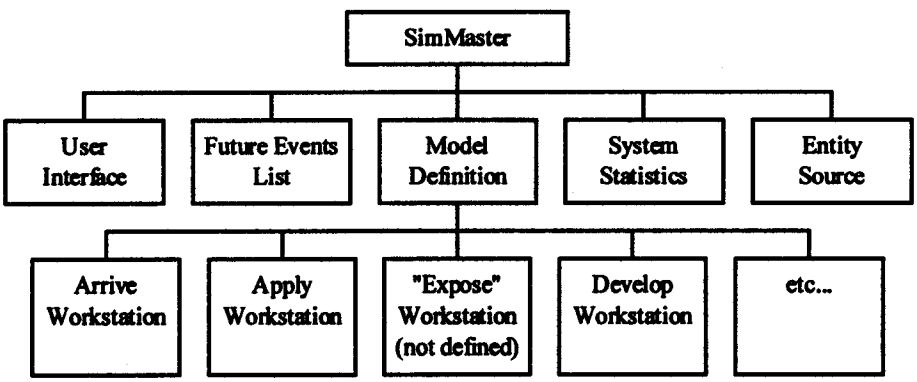

Figure 5.3: Simulation Module Major Object Relationships of NCSOS

Once the model has been defined, it is compiled. Of course, at compile time, the name or behavior of the "Expose" workstation is not known. This object will be created at Simulation Module runtime.

5.2.7 The Target System

Once again, the objective of an independent analyst is to create a Java object (a Java class) that can seamlessly interact with the Simulation Module when the modeler executes a simulation run. The imported object will simulate the behavior of an "Expose" workstation in a semiconductor fabrication system.

A semiconductor wafer manufacturing process was selected to model because it has been, traditionally, a difficult process to accurately model. The entities (wafers) have a complex and diverse routing through the process and include many re-entrant behaviors.

5.2.8 Expose Workstation Behavior

If the analyst is not familiar with the characteristics of an "Expose" workstation, don't worry, this is a theoretical system; there are no wrong answers. The analyst is encouraged to read the examples and to use his/her imagination to simulate what an "Expose" workstation might do.

The exposure process in semiconductor manufacturing is very similar to exposure of photography paper on an enlarger. The UV light-sensitive photoresist is applied to the

surface of the wafer in the "Apply" workstation. Then, in the "Expose" workstation a "mask" is put on top of the wafer and the wafer is exposed to UV light for some period of time. Afterwards, the wafer is passed to the "Develop" workstation for further processing.

5.2.9 The Discrete-Event Simulation Scenario

Hopefully, the analyst is familiar with the concept of discrete-event simulation. As simulation time is incremented in the model, events are scheduled in the future events list, and at each "event time," the event at the front of the list is "processed." For instance, if the next event to be processed is an "arrive" event on the Expose workstation, then dialog between the SimMaster and the Expose Workstation might be something like this:

SimMaster: "Hey, Expose, here is wafer #3342. It just arrived at your front door."

Expose Workstation: "Ok, thanks. Please schedule a "load" event immediately.

SimMaster: "Hey, Expose Workstation, if you can, load a wafer.
Expose Workstation: "Ok, thanks. I've loaded wafer #3342. Please schedule an "unload" event for this wafer in 5 minutes.

So this communication continues event after event. Typically, each time a workstation receives an event, it does some processing, then schedules at least one event with the SimMaster. This is how the simulation continues.

The NCSOS environment is no different than many standard environments. Every time a workstation receives an event, it processes the event, and schedules at least one

more event. The simulation continues until there are no more events, or until the simulation run time has been reached.

5.2.10 Class Structure

The hierarchy of the classes that the analyst must be aware of is given in Figure 5.4.

Figure 5.4: Important Environment Interface Classes.

5.2.10.1 The SimMaster Class

The SimMaster class is responsible for managing the simulation, scheduling events, and sending each workstation a message when an event occurs at that workstation. Most of this communication is hidden from the analyst, but when an event is scheduled, the SimMaster class will eventually schedule the event by creating a new instance of SimEvent, and inserting it into the future event list.

The messages that are useful:

1. **SimMaster.getSimTime()** – A static method that returns a "float" that is the current simulation time.
2. **SimMaster.printEvent(String s)** – A static method that allows you to send a string to the trace display in the user interface. Each event will automatically be output to the trace display, but if you want to display something additional, just send a string to SimMaster.printEvent().

5.2.10.2 The Workstation Class

The Workstation class is the superclass (or parent) of all specific workstation classes. A specific instance of Workstation is never created. Rather, the Workstation class is used as a basis of the structure and behavior of all specific workstation classes.

The analyst must "extend" the Workstation class when defining the "Expose" class (The workstation name can be any valid class name). Most of the communication that the specific workstation does with the simulation environment is accomplished by using the Workstation parent class.

Important methods for the Workstation Class:

1. **arrive(Entity ent)** – If a child workstation does not define an "arrive" method, the workstation will just schedule a "depart" event from this workstation immediately.
2. **depart(Entity ent)** – If a child workstation does not define a "depart" method, the workstation will schedule an "arrive" event to the NEXT

workstation immediately. (The next workstation is obtained from the entity itself.) It is suggested that the analyst not define a "depart" method, and just let the Workstation class take care of moving the entity to the next workstation.

3. **scheduleEvent(Float dt, Entity ent, String method)** – Schedules an event using the parameters that are sent. So, when the analyst wants to schedule an event from the "Expose" workstation, a message is sent to the super object with the appropriate parameters.
    - "dt" is the delay of the event. This would be the "delta time" in the future that the event will occur.
    - "ent" is the Entity that is associated with the event.
    - "method" is the name of the next event.
4. **scheduleEvent(Float dt, String method)** – This is another form of the scheduleEvent() method. It is for events that don't have a related Entity. Schedules an event using the parameters that are sent.
5. **reportFinalStatistics()** – This method is called at the very end of the simulation run. Any statistical reports that are to be displayed as part of the output of the run should be entered in this method. See the examples for more info.

Of course, the definition of the Workstation class is more complex than these few methods. However, it is not necessary to understand, or even be aware of the

implementation details to use the functionality of the Workstation class. This is an excellent example of OOP "encapsulation" at work. The Java code for the Workstation class is located in Appendix A.

5.2.10.3 The Entity Class

The entity class represents the wafers flowing through the system. The entities are actually complex objects, as they keep track of their own statistics, they know their routing through the system and their processing time at each step. Most events are associated with an entity, so whenever an Entity-related event is processed or scheduled, the entity should be included in the message.

Useful methods for the Entity Class:

1. **getProcessingTime()** – Returns a "float" that is the processing time for this entity at this step in the processing sequence. You may choose to use this processing time, or determine your own when scheduling future events.
2. **getId()** – Returns an "int" that is the unique ID of this entity.

5.2.11 Environment Restrictions

This simulation environment is a prototype, and is limited in it functionality and usefulness to model systems outside of the domain considered in this research. Accordingly, there are several significant restrictions that need to be mentioned.

The "Expose" workstation is limited in the external objects it can interact with. The analyst can not define other workstation "support" classes at this time. Everything the "Expose" workstation does must be done internally, or with the use of:

1. The Interface objects mentioned above.
2. All standard Java classes from the JDK 1.1.
3. All classes from the Java Generic Library (jgl) package. These are public domain "support" classes that are included when downloading the NCSOS software. For detailed information on these useful classes, see http://www.objectspace.com.

5.2.12 Expose Workstation Example

Probably the easiest way to get a feeling of how the interface to the Simulation Module works is by studying an example of the code for an "Expose" workstation. In this example, there is a simple one-server workstation. When an entity enters, the "arrive" method puts it into a queue, and schedules a "load" event. Then a method, "load" is defined that looks to see if the server is available. If it is, then the entity is loaded on the server, and an "unload" event is scheduled in the future. When the "unload" event occurs, the server is freed, and a "depart" event is scheduled. These are the events that occur on this workstation for each entity that arrives to this workstation.

Of particular importance is the fact that the analyst can create any event that is desired. There could be an event called "someStrangeEvent()" that no other object in the

environment "knows" about. The analyst must just ensure that when the "Expose" workstation receives such a method, it "knows" how to deal with it.

The Java code for the ExposeWorkstation class is found in Figure 5.5 and Appendix C.

```
import jgl.DList; // a linked-list from the Java Generic Library

// the name of the class is upto the analyst.
// it is mandatory to extend Workstation and implement RemoteWorkstation
class ExposeWorkstation extends Workstation implements RemoteWorkstation {

    // Data fields of this class
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a double linked-list from the jgl package
                         // could use somethink like an Array also.
    float inUse; //status of server
    float startService; //start of service

    //Constructor
    ExposeWorkstation() {
        // init the Workstation object
        super();
        // give this a name
        workstationName = "Remote Expose Workstation";
        // number of servers in this workstation
        servers = 1;
        // create a queue to store entity waiting for processing
        queue = new DList();
        // status variable
        inUse = 0;
        // time marker
        startService = 0;
    }
    // Overrides the arrive method in the Workstation class.
    //Needed if any processing is going to be done at this workstation.
    public void arrive(Entity ent) {
        // put entity into queue
        queue.pushBack(ent);
        // schedule a load event in "0" time delay
        super.scheduleEvent(0,"load");
    }
    // Define a "load" event.
    // the parameter Object is a dummy.  must be used if the event
    // is not Entity-related.
    public void load(Object obj) {
        if (servers == 0) {
            SimMaster.printEvent("Expose Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Expose Queue.");
            return;
        }
```

Figure 5.5: (Continued)

```
        // Otherwise, load the entity
        // decrement the servers avaiable.
        servers--;
        // mark start time
        startService = SimMaster.getSimTime();
        // get the first entity in queue
        Entity ent = (Entity)queue.popFront();
        SimMaster.printEvent("Successfully loaded Entity: #"+ent.getId());
        // get the processing time from the entity
        float dt = ent.getProcessingTime();
        // schedule the unload event
        super.scheduleEvent(dt,ent,"unload");
    }
    // define the "unload" event
    // notice this is an "entity-related" method.
    // the related entity is the parameter
    public void unload(Entity ent) {
        // increment the servers available.
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        // check the queue, non-entity related
        super.scheduleEvent(0,"load");
        //schedule a depart event immediately, entity related
        super.scheduleEvent(0,ent,"depart");
    }

    // No need to define the "depart" event here.  Just let the super object
    // take care of sending the entity to the next workstation.

    public void reportFinalStatistics() { // called at the end of the run
        super.makeStatisticsWindow(workstationName+" Final Statistics");// (mandatory
line)
        //report the collected stats in the following lines.
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

Figure 5.5: ExposeWorkstation Example Java Code.

5.2.13 Interface Rules

The "rules" of the interface are summarized below:

1. The "Expose" workstation must extend Workstation and *implement* (a Java keyword) RemoteWorkstation. This guarantees a basic behavior by Workstation class and allows the Simulation Module to dynamically create an instance of an object that implements the RemoteWorkstation interface. The name of the class can be any valid Java class name.

2. If any processing is to be done, the super Workstation class method "arrive()" must be overridden. This "intercepts" the entity, and allows the analyst to manipulate the entity as it passes through the workstation.
3. Typically, at each event, at least one succeeding event must be scheduled by asking the super Workstation object to schedule an event. If no future event is scheduled, the current process will come to a halt. For example, if a "load" event occurs, but no "unload" event is scheduled, the entity will continue to occupy the workstation indefinitely. An event is scheduled by sending the super object a method scheduleEvent().
4. The analyst can implement any new event that is desired. This is accomplished by defining a new method in the class (someNewEvent(), for example). The method has no return value and must take either Entity or Object as its parameter. If it is an Entity related event, such as "unload", where the event is directly related to a specific entity, then the event should use Entity as the parameter. Otherwise, if the event is not related to a specific entity, then the parameter should just be a dummy Object. This is undesired, but a current limitation of Java. It would be more accurate for a parameter-free method, but this was not possible with the current release of Java.
5. The analyst will schedule the newly created events in any order, at any time delay. It is imperative that if an event is scheduled, then it can be handled by the "Expose" workstation. If an event arrives to the "Expose"

workstation that is not defined, an error will occur and the simulation will terminate.

6. The analyst can implement as many different internal events as desired.
7. The last event at the "Expose" workstation should be to schedule a depart event. This will allow the super Workstation object to pass the entity to the next workstation. This is accomplished by scheduling a "depart" event. The syntax would be: super.scheduleEvent(0, ent, "depart"). This would be the last call in the processing of an entity at this workstation.
8. The "Expose" workstation must implement the "reportFinalStatisics() method. See the example for the syntax. The analyst is free to keep any statistics that are desired. This would be accomplished by defining instance variables and recording the desired statistics as the simulation run progresses. For example, this may be server utilization, number of entities processed, time entities spend in queue or in service, etc. At the end of the simulation run, the Simulation Module will send the reportFinalStatistics() message. At this time, the statistics that are to be reported should be composed into a String and sent to the super object with the message super.printStat(String). The super Workstation object will create a window with the corresponding information. One statistic should be sent for each printStat() call. There is no need to worry about formatting, as the Workstation will take care of it.

5.2.14 Downloading Necessary Software

In order to build a workstation object, the independent analyst must do the following:

1. Download and install the Java JDK, version 1.1 or higher. Available from: http://java.sun.com/products/jdk/1.1/
2. Download and install the NCSOS Software. Available from: http://colvin.ie.orst.edu/ncsos/install/

5.2.15 Installing and Using Necessary Software

Before the NCSOS software can be used, the analyst must have installed the JDK version 1.1 or later.

WARNING: All the development of the NCSOS software was done on the Windows NT operating system. Nothing has been compiled or tested on the UNIX operating system. (Java byte code claims to be "Platform-Independent," but it has not been verified for this project).

The downloaded file (ncsos.exe) is a self-extracting file. It should be extracted into a working directory ("c:\ncsos\" for example).

The software included is:

- *.class - The class files needed to compile the "Expose" workstation file that will be created.

- \src*.java - The Java code used to compile the .class files. These files should not be needed for object development, but the code as been included for inspection if the analyst desires. This includes the example ExposeWorkstation.java file.
- \jgl\ directory - The Java Generic Library package. Useful classes for queues, etc. See more information from http://www.objectspace.com/

The analyst will create the "Expose" workstation .java file. It should be located in the "ncsos" directory. To compile the file, the command would be:

```
C:\ncsos\>javac ExposeWorkstation.java
```

This is following standard procedure for using the Java JDK 1.1. Of course, the name of the .java file can be any valid Java class name.

Once the file has been compiled, the .class file needs to be posted on an ftp server. The location of the file should be accessible for an "anonymous" ftp connection.

5.2.16 Testing Object Behavior

A simplified version of the Simulation Module has been included with this software. To start the Simulation Module, at the command prompt, type:

```
C:\ncsos\>java SimApplication
```

This Java application will send one Entity through the "Expose" workstation, every 0.5 hours. The entity has a processing time of 1 hour on the workstation, so entities

arrive twice as fast as they exit. Each event will echo to the screen as it is processed. This should help the analyst test the object.

When running the Simulation Module, enter the class name of the object (For example, "ExposeWorkstation"), and the ftp URL (For example, "ftp://imesupport.ie.orst.edu/ncsos/"). When these are valid, the Simulation Module can be run.

5.2.17 Posting Objects in the Public Domain

Once the object has been compiled, tested and posted on an ftp server, alert the intended modeler with the name and location of the object. The full version of the Simulation Module (using the entire model) can then be run using this representation of an "Expose" workstation.

5.2.18 A Final Word From the Author

This is a prototype simulation environment. It has been designed for use in the very specific domain that is described in this research. The use of this environment outside of this narrow domain has not been tested, and unforeseen conflicts may be encountered. The objective of this research was to determine if this methodology of "Network-Centric Simulation Modeling" is feasible and worth more effort. There has been a deliberate intent to keep this system as simple as possible during this "first try".

If you have any suggestions, feedback, advice or input please don't hesitate to contact the author.

CHAPTER 6. ENVIRONMENT APPLICATION

6.1 Introduction

In this chapter, the simulation environment is applied to a target semiconductor fabrication system. The purpose of this task is to show how the environment can be applied, and how its network-centric functionality can use resources on the Internet.

First, the general model specification is presented, followed by a brief description of a semiconductor fabrication system. Next, the specific behaviors of two different Expose workstations are introduced. These workstations are then implemented as the network-centric objects that will be used by the environment to perform an example of network-centric simulation modeling. Finally, the results from the simulation runs are presented.

6.2 General Model Specification

There are three basic object locations where the specific model parameters are defined. It is the combination of these three components that create the unique simulation model of the system under investigation.

First, the SimModel object defines specific workstation objects, workstation layout and identifies the workstations that will report statistics at the end of the simulation run. For a detailed description of the SimModel class definition, see section 6.4.

Second, the specific workstation objects define the processing operations that are unique to each workstation. In the NCSOS environment, each workstation acts as an independent location that receives wafers, performs processing on the wafers, and releases wafers to the next workstation. In the current implementation of NCSOS, all workstations are locally defined objects except for the single remote workstation. However, because the remote workstations are created according to the NCSOS standard, they will seamlessly interact with the simulation environment in much the same way as the locally defined workstations. See Appendix B for the locally defined workstation class definitions and Appendix C for an example of a remote workstation class definition.

Finally, the Entity object defines the routing, specific workstation processing times and statistic collection for the wafers that flow through the system. One of the powerful aspects of the NCSOS environment is the ability for the wafers to "know" their processing times, workstation routing and current step in the processing sequence. This facilitates the creation of multiple wafer types that contain unique processing times and routing sequences. See Appendix A for the class definition of the Entity object.

6.3 Description of Target System

Once again, the system under consideration is a simplified semiconductor fabrication system. For a more detailed description of the system, see chapters 1, 3, and 4.

Wafers enter the system, and are processed on workstations in a predefined sequence. This sequence does not necessarily follow a linear flow, but may be varied greatly depending upon the wafer's specifications. One unique characteristic is the reentrant flow; wafers may visit the same workstation many times. Two of the primary measures of performance for such a system would be the wafer throughput and cycle time.

In the application considered here, the emphasis is placed on the behavior and performance of the Expose workstation. Therefore, the primary focus of this example will center on the modeling of the Expose workstation.

The following list of assumptions were applied to the system for this model:

1. Line loading (WIP) will be set to 20 units. Only when a wafer exits the system, will another be introduced into the system.
2. All workstations except the Expose workstation will have the following characteristics:
    - No load/unload delay
3. Processing time at each workstation will be a uniformly distributed random variable between 2 and 3 hours.
4. Simulation run length will be 10,000 hours.
5. No transient state will be observed.

6.4 Description of Expose Workstations

Recall from chapter 4 that the function of the Expose workstation is to expose UV light onto the surface of a semiconductor that has had a photoresist material applied to its surface.

In this model, two different Expose workstation designs were considered. First, a single-server workstation was modeled, followed by a dual-server configuration. The primary question that was asked for this model was: Does one workstation configuration perform better than the other in this system?

The single-server workstation was created with the following specifications:

1. Name: ExposeS1000
2. Number of Servers: 1
3. Processing Time: Dependent on Wafer.
4. Load Time: Uniformly Distributed, 10 - 20 Minutes.
5. Unload Time: Uniformly Distributed, 10 - 20 Minutes.

The dual-server workstation was created with the following specifications:

1. Name: ExposeS2000
2. Number of Servers: 2
6. Processing Time: Dependent on Wafer.
3. Load Time: Uniformly Distributed, 30 - 40 Minutes.
4. Unload Time: Uniformly Distributed, 40 - 60 Minutes.

The workstation objects representing these machine specifications were implemented using the documented standard presented in chapter 5. The result of this implementation was two software objects (Java class files) that represent the respective

workstation design. For the Java code used to implement these workstations, see Appendices D and E.

6.5 Simulation Model and Execution

Once the software objects representing the workstations were compiled, they were placed on an ftp server with public access from the Internet. These objects were now accessible to the public domain, including the NCSOS environment, regardless of the location of the computer on which any of the software was located.

The simulation model was defined in the SimModel class, which was presented in chapter 4. A more detailed description of the model is presented here. For a full listing of the Java source code for SimModel, see Appendix A.

To create a simulation model using the NCSOS framework, there were three distinct steps. First, the instance variables were declared. Second, the model constructor was created. Finally, it was determined which workstations will collect statistics over the simulation runs.

Declare Instance Variables. In this step, a variable for each of the workstations was declared. The name of the variable can be any valid Java variable name. The Java modifier for each variable is "private," as this restricts any unwanted modification by external objects. In addition to a variable for each of the workstations, several other necessary variables were defined. These were the arrival location, terminate location, the

system statistics variable, and the workstation layout variable. The instance variable definition for the semiconductor fabrication system is seen in Figure 6.1.

```
// Workstation Variable Declaration
    private ArrivalLocation arrival;
    private ApplyWorkstation workstation1;
    private RemoteWorkstation workstation2;
    private DevelopWorkstation workstation3;
    private BakeWorkstation workstation4;
    private DepositWorkstation workstation5;
    private EtchWorkstation workstation6;
    private TestWorkstation workstation7;
    private TerminationLocation terminate;
// Other Needed Declarations
    private SystemStatistics systemStatistics;
    private DList workstationLayout;
```

Figure 6.1: Workstation Variable Declaration.

Create Model Constructor. The next step was to create a new instance of each workstation and assign it to its corresponding variable. This was done in the constructor of the SimModel class, which is executed upon creation of the SimModel object. Also in this step, each workstation was assigned a location in the workstation layout. This facilitates the routing of the entities through the system. Figure 6.2 presents the Java code used to accomplish these tasks.

```
SimModel() {   // Constructor
        //needed objects
        workstationLayout = new jgl.DList(); // the list of Workstation Objects, in order
        // add arrival Location
        arrival= new ArrivalLocation(); // interarrival time
        // define local workstations, must be in order.
        workstation1 = new ApplyWorkstation();
        //define the remote workstation
        workstation2 = (RemoteWorkstation)RemoteLoader.loadWorkstation();
```

Figure 6.2: (Continued)

```
        if (workstation2 == null) {
            SimMaster.printEvent("Did NOT Successfully Load Workstation\n");
            SimMaster.printEvent("Please push \"Reset\" and enter
             a valid Name and URL\n");
        } else {SimMaster.printEvent("Successfully Loaded Workstation\n");}
        //then the rest of the local workstations
        workstation3 = new DevelopWorkstation();
        workstation4 = new BakeWorkstation();
        workstation5 = new DepositWorkstation();
        workstation6 = new EtchWorkstation();
        workstation7 = new TestWorkstation();
        // Finally Add Terminate Location
        terminate = new TerminationLocation();
        // Build workstationLayout
        workstationLayout.pushBack(arrival);//put ws in list
        workstationLayout.pushBack(workstation1);//put ws in list
        workstationLayout.pushBack(workstation2);//put ws in list
        workstationLayout.pushBack(workstation3);//put ws in list
        workstationLayout.pushBack(workstation4);//put ws in list
        workstationLayout.pushBack(workstation5);//put ws in list
        workstationLayout.pushBack(workstation6);//put ws in list
        workstationLayout.pushBack(workstation7);//put ws in list
        workstationLayout.pushBack(terminate);//put ws in list
    }
```

Figure 6.2: Simulation Model Constructor.

Identify Workstations to Collect Statistics. Finally, it was specified on which of the workstations statistics would be collected. It may be that there is no need to collect statistics on one or several workstations in the model. Therefore, for each of the workstations where statistics are desired, a specific message must be sent requesting that the statistical information be reported at the end of the simulation run. This was a very straightforward task as Figure 6.3 indicates.

```
// Final Statistics
    public void reportFinalStatistics() {
        workstation1.reportFinalStatistics();
        workstation2.reportFinalStatistics();
        workstation3.reportFinalStatistics();
        workstation4.reportFinalStatistics();
        workstation5.reportFinalStatistics();
        workstation6.reportFinalStatistics();
        workstation7.reportFinalStatistics();}
```

Figure 6.3: Collection of Statistical Workstation Data.

Finally, with the model fully defined, a series of 5 simulation runs was performed for each of the workstation objects presented above. This entailed starting the NCSOS Simulation Environment, specifying the Expose workstation name and URL, and entering the simulation run length.

6.6 Results

The results from the simulation runs appear in Table 6.1.

Table 6.1. Simulation Run Results.

		Model S1000		Model S2000	
Measure of Performance	Run	Ave.		Ave.	
Cycle Time (Hours)	1	59.5790		50.3879	
	2	59.4141		50.4172	
	3	59.5520		50.4583	
	4	59.5331		50.2802	
	5	59.5668		50.3849	
Ave		59.5290		50.3857	
Throughput (Hours)	1	0.3348		0.3959	
	2	0.3357		0.3958	
	3	0.3349		0.3954	
	4	0.3350		0.3969	
	5	0.3348		0.3960	
Ave		0.3350		0.3960	
				Server 1	Server 2
Server Utilization	1	0.9997		0.7839	0.8501
	2	0.9995		0.7825	0.8351
	3	0.9994		0.7872	0.8302
	4	0.9996		0.7858	0.8497
	5	0.9996		0.7911	0.8500
Ave		0.9996		0.7861	0.8430

The results from this simulation model indicate there is a statistically significant improvement in cycle time and throughput using the S2000 workstation configuration over the S1000 workstation. See Appendix F for the hypothesis test results.

Additionally, to validate the NCSOS environment, a simulation model was constructed using ProModel. The ProModel simulation model duplicated the NCSOS model using the single server Expose workstation configuration. Of course, ProModel does not employ a network-centric approach. However this task was done to validate the measures of performance obtained from the NCSOS simulation runs. There was no statistical difference in the results obtained from both environments. See Appendix G for a comparison of the ProModel and NCSOS results.

6.7 Conclusions

This chapter has documented an example of how the NCSOS environment can be used to simulate a system using a network-centric methodology. While this was not a particularly complex or sophisticated example, it shows the potential power of the network-centric approach. The true power, however, will not be realized until this prototype has been developed into a more robust and complete simulation environment. Future research opportunities using the network-centric approach are discussed in the next chapter.

CHAPTER 7. FUTURE RESEARCH

7.1 Introduction

This research has documented the NCSOS environment from conception, to a functional prototype with a documented standard. It must be realized that the environment is just that; a prototype. The modeler is severely limited in the systems that can be modeled, and the detail that can be incorporated into a model. However, this research has opened the door to a new methodology for the simulation of systems: the network-centric paradigm.

This chapter presents eight specific areas that have been identified as worthwhile developments and enhancements to the NCSOS environment. Some of the areas are relatively straightforward and have been accomplished in other environments. The remaining areas are still undeveloped by any known simulation software. The following nine sections present possible future research regarding network-centric simulation.

7.2 Increased Functionality and Robustness of Workstation Objects

In the current environment, the remote workstations are limited in the actions that can be simulated. For instance, there is no functionality for periodic, unplanned failure. The remote workstations are also constrained to use only specified external objects. They are not free to implement support classes that would increase their functionality and allow

the analyst to more accurately abstract a physical system. The environment would benefit if a more flexible standard for remote workstations were to be implemented.

7.3 Develop New Network-Centric Objects

In the current environment, only objects that represent workstations are implemented as network-centric. An intuitive development would be to develop other network-centric objects. These would include such objects as entities, transportation equipment, manual operators, and informational and logic objects. The challenge to implementing such objects as network-centric is their need to be independent and encapsulated, so that they may be developed according to the standard in a remote environment. However, with such a system, the simulationist could become an "assembler" of simulation models rather than a "creator."

7.4 Improved Statistics Module and Output Reporting

The existing environment has a basic subsystem for collecting specifically defined statistics. Clearly, to become a useful environment capable of asking complex "what-if" questions, a more complete statistics collection module is needed. In addition to the collection of measures of performance, a system for statistics reporting is needed to make the output data from a simulation run effortless.

7.5 Facilitate Model Building Capabilities

While a substantial effort was made in the existing environment to make model building modular and intuitive, it is still a very cryptic and programming-intensive task. Much could be done to improve this area of the environment, including the evaluation of a method to eliminate the need to compile the model. An even more extreme enhancement would be to create a graphical model building capability. This would facilitate user-friendly model building capabilities.

7.6 Develop Model Input Data Module

In the existing environment, there is no central collection of model input data. It is distributed over several class definitions, and is cumbersome to implement changes in the input to a simulation model. There is a need to develop a subsystem to define and store the model input data in an intuitive, concise structure.

7.7 Develop Random Variable Generation Module

This is a very straightforward enhancement. Many other simulation environments have very sophisticated random number modules that can accommodate many random variable distributions. This enhancement would be time consuming, but would also add important functionality to the environment.

7.8 Increase Efficiency and Execution Speed of Model

Even with the relatively small size and sophistication level of the existing environment, the execution speed of the model is noticeably slow. Some of this can be attributed to the infancy of the Java language. It is anticipated that new Java-related technologies will increase the efficiency of large Java applications, but some efficiency improvements are possible in the coding of the existing environment by using less sophisticated data structures and user interface components.

7.9 Convert the Application to a Web "Applet"

The current capabilities of Java did not allow "applets" the functionality needed to implement this simulation environment. However, as Java evolves, it is expected that the functionality of Java will increase, especially in the area of larger and more robust "applets." The task of adding this functionality to the environment is anticipated to be a relatively straightforward undertaking.

7.10 Develop Graphical Runtime Capabilities

Graphical simulation environments are especially useful when trying to sell simulation recommendations to non-simulationists. There are times, however, that a graphical interface can assist the modeler in validation and verification of a model. This

would be a very time consuming enhancement, perhaps best accomplished by an experienced graphical programmer.

7.11 Summary

In conclusion, this prototype is a functional, yet limited simulation environment that incorporates the innovative network-centric methodology. This chapter has outlined nine of the further research opportunities to enhance and improve the functionality of the existing prototype environment.

CHAPTER 8. CONCLUSIONS AND RECOMMENDATIONS

The primary goal of this research was to investigate the concept of network-centric simulation. To achieve this goal, four distinct research objectives were established (see Chapter 1). This chapter first discusses the conclusions from this research in the context of these objectives. Finally, this chapter summarizes the final recommendations of this research.

8.1 Summary of Research Objectives

8.1.1 Specification

The first objective of the research was to develop the specifications for the project. The first specification developed was for the discrete-event simulation environment. This closely followed a specification for a typical discrete-event simulation "engine." The second specification was related directly to the concept of network-centric simulation. Here, the concepts were developed to use the Internet as a resource when building simulation models. This specification reflected an object-oriented approach of using independent, encapsulated objects to represent simulation components. The third specification defined the requirements for the system to be modeled. The target system selected for this research was a semiconductor fabrication system. The requirements for this system were defined, reflecting the domain-specific characteristics of these complex

systems. Fourth, the specification for a documented standard was developed. In order for a network-centric environment to be of any use, simulation analysts must be able to use the standard to develop simulation objects in a independent, isolated environment. The standard documentation allows for distributed use of the environment. Finally, five design goals were identified for the implementation of the simulation environment.

8.1.2 Implementation

The second objective of the research was to implement the specification. The object-oriented programming language, Java, was selected as the implementation language because of its ease of use and network-programming capabilities. First, a class hierarchy was developed methodically and structurally to meet the discrete-event simulation specifications. Next, the basic functionality of importing remote objects was completed, then, a more sophisticated environment was developed to include user interface components and statistics collection. Finally, the model specifications were met using the network-centric simulation framework to implement the target system model.

8.1.3 Standard Development

The third objective was to gather and organize the interface of the environment into a documented standard that a simulation analyst could use to independently develop remote simulation objects. This documentation included a description of the environment, a description of important interface object behavior, a sample remote object

definition and a standard for implementing remote objects. This document and the accompanying software were posted on a public domain Web page for easy and rapid distribution to simulation analysts.

8.1.4 Environment Application

Finally, the environment was applied to a target system. The purpose of this task was to show how the environment can be used to model a system using remote, network-based objects representing semiconductor wafer-processing workstations. Two specifications for Expose workstation behavior were created, then implemented according to the NCSOS Standard documentation. The objects were placed on the Internet, and simulation runs were executed using these objects to represent the Expose workstation in the semiconductor simulation model.

8.2 Final Recommendations

There are three final recommendations drawn from this research.

1. The advantages of object-oriented simulation software continue to far outweigh the disadvantages. While some ease of use is compromised by a non-graphical or non-scripting interface, the flexibility and ability to ask very abstract "what-if" questions is worth the extra effort.
2. The network-centric computing model is quickly becoming feasible with the emergence of Internet technologies and tools such as Java. The near

future will bring a variety of computer applications that are no longer restricted to a desktop-centric environment. These new generation applications will integrate and incorporate a variety of resources and services that are located on the Internet.

3. The combination of object-oriented programming and network-centric computing can be combined into a feasible network-centric simulation methodology. This method of simulating systems would allow a common ground between simulationists of similar systems to share and incorporate Internet-based simulation objects.

BIBLIOGRAPHY

Ball, Pete, and Doug Love. (1994) Expanding the Capabilities of Manufacturing Simulators Through Application of Object-Oriented Principles. *Journal Of Manufacturing Systems*, 13-6: 412-23.

Banks, Jerry, and John S. Carson (1984) *Discrete-Event System Simulation*, Prentice-Hall, Inc., New Jersey.

Beaumariage, T. G., (1990) "Investigation of an Object-Oriented Modeling Environment for the Generation of Simulation Models," Ph.D. Dissertation, School of Industrial Engineering and Management, Oklahoma State University, Stillwater, Oklahoma.

Budd, Timothy, (1996) *An Introduction to Object-Oriented Programming*, Addison-Wesley, Reading, Massachusetts.

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.*

Chang, W. and L. R. Jones, (1994) Message-Oriented Discrete Event Simulation, *Simulation*, 63:2, 96-100

Cox, Brad J., (1986) *Object Oriented Programming: An Evolutionary Approach*, Addison-Westly, Reading, Massachusetts.

Dietz, Dan, (1996) Java: a new tool for engineering, *Mechanical Engineering*, April, 68-72.

Evans, John B.(1988) *Structures of Discrete Event Simulation*, Ellis Horwood Limited, New York.

Eldredge, David L., John D. McGregor, and Marguerite K. Summers (1990) Applying the object-oriented paradigm to discrete event simulation using the C++ language, *Simulation*, 55: 83-91

Floyd, Michael (1989) A Class Act. *Dr. Dobb's Journal*, April:58-64.

Flynn, Jim, and Bill Clarke (1996) How Java Makes Network-Centric Computing Real, *Datamation*, March, 42-3.

Glassey, C. R., and S. Adiga (1990) Berkeley Library of Ojbect for Control and Simulation of Manufacturing (BLOCS/M), *Applications of Object-Oriented Programming*, Addison-Wesley, Reading, Massachusetts.

Goldberg, A., and D. Robson, (1983) *Smalltalk-80: The language and Its Implementation*, Addison-Wesley, Reading, Massachusetts.

Graybeal, Wayne J., and Udo W. Pooch.(1980) *Simulation: Principles and Methods*, Winthrop Publishers, Inc., Cambridge, Massachusetts.

Hood, Sarah Jean, Amamoto, Amy E. B., Vandenberge, Antonie T., (1989) A Modular Structure For A Highly Detailed Model of Semiconductor Manufacturing, *Proceedings of the 1989 Winter Simulation Conference*, 811-817.

Miller, David J., (1994) The Role of Simulation in Semiconductor Logistics, *Proceedings of the 1994 Winter Simulation Conference*, 885-891.

Miller, John A., Rajesh S. Nair, Ahiwei Zhang, and Hongwei Zhao (1996) JSIM: Java-Based Simulation and Animation Environment, *Proceedings of the 1996 Winter Simulation Conference*

Ozden, Mufi H., (1991) Graphical programming of simulation models in an object-oriented environment. *Simulation*, 56:2, 104-116.

Pegden, C. Dennis, Robert E. Shannon, and Randall P. Sadowski (1990) *Introduction to Simulation Using SIMAN*, McGraw-Hill, Inc., Hightsown, New Jersey.

Pooch, Udo W., and James A. Wall. (1993) *Discrete Event Simulation: A Practical Approach*, CRC Press, Boca Raton, Florida.

Pritsker, A. Alan B.(1986) *Introduction to Simulation and SLAM*, Systems Publishing Corporation, West Lafayette, Indiana.

Pritsker, A. Alan B.(1990) *Papers, Experiences, Perspectives*, Systems Publishing Corporation, West Lafayette, Indiana.

Rosenberg, Ronald C., Joseph Whitesell, and John Reid, (1992) Extendible simulation software for dynamic systems, *Simulation*, 58:3 175-183.

Schmidt, J. W., and R. E. Taylor. (1970) *Simulation and Analysis of Industrial Systems*, Richard D. Irwin, Inc., Homewood, Illinois.

Semich, Bill and David Fisco (1996) Java: Internet Toy or Enterprise Tool?, *Datamation*, March: 28-29.

Shannon, Robert E. (1975) *Systems Simulation: The Art and Science*, Prentice-Hall, Inc., New Jersey.

APPENDICES

APPENDIX A: Java Environment Class Definitions

ArrivalLocation.java

```
import java.lang.reflect.Method;

class ArrivalLocation extends Workstation {  // Entry Point

    // Data fields of this class
    private float dt = 0; //used to schedule events
    private int servers; // number of servers in this workstation

    //Constructor
    public ArrivalLocation() {
        super();
        workstationName = new String("Arrival Location");
    }
    // Methods
    public void arrive(Entity ent) { // just schedule arrival to first workstaion
        // set any stats that are needed
        ent.setCreationTime(SimMaster.getSimTime());
        SimMaster.updateWIP(1); // increate the WIP
        // let Workstation do the rest
        super.arrive(ent);
    }
}
```

ClassSaver.java

```
import java.net.*;
import java.io.*;

class ClassSaver {

    private static String codebase = null;
    private static String fileName = null;
    private static byte[] data = null;

  public ClassSaver(){
  }
  public static void saveRemoteClass(String className,String cb) {

    fileName = className + ".class";
    codebase = cb;
    String thisURL = codebase+fileName;
    data = null;

    // load the data
    try {
      System.out.println("Connecting to URL: " + thisURL);
      URLConnection u = new URL(thisURL).openConnection();
      DataInputStream in = new DataInputStream(u.getInputStream());
      data = new byte[u.getContentLength()];
      in.read(data);
      in.close();

    // save the data
        File target = new File(fileName);
        if (target.exists()) {target.delete();}
        FileOutputStream out = new FileOutputStream(target);
        out.write(data);
        out.close();
        System.out.println("Finished Creating file: " + fileName);

    }   catch (IOException e) {
        SimMaster.printEvent("Did NOT Successfully Load: "+thisURL+"\n");
        SimMaster.printEvent("Please push \"Reset\" and check the Name and URL again\n");
        System.out.println("Caught IOException trying to read URL: " + thisURL);
    }
  }
}
```

Creator.java

```
import java.lang.reflect.Method;

class Creator {  // creates and schedules new entities

    // data fields
    private float dt;
    private Entity newEntity;
    private Workstation arrivalLocation;
    private SimMaster simMaster;

    Creator(SimMaster sm) { //Constructor
        simMaster = sm;
        arrivalLocation = sm.getArrivalLocation();
    }
    public void newEntity() {
        newEntityIn((float)0);
    }
    public void newEntityIn(float time) {
        dt = time;
        newEntity = new Entity();
        // schedule arrival to ArrivalLocation in dt
        Class[] args = new Class[1];
        args[0] = newEntity.getClass();
        Method msg = null;
        try {
            msg = arrivalLocation.getClass().getMethod("arrive",args);
        }  catch (NoSuchMethodException e) {
            System.out.println("From Creator: Caught
NoSuchMethodException"+e.getMessage());
        }
        simMaster.scheduleEvent(new SimEvent(dt,newEntity,arrivalLocation,msg));
    }
}
```

Entity.java

```
import jgl.DList;
import jgl.DListIterator;

class Entity {

    // Data fields of this class
    private static int idCounter = 1; // the unique id of this entity
    private int id;
    private DList sequence; //sequence of entity through fab, with pt
    private SequenceStepInfo currentStep = null;
    private DListIterator i;
    private Workstation nextWorkstation;
    private EntityStatistics stats;
    // Constructor
    Entity() {
        id = idCounter;
        stats = new EntityStatistics();
        sequence = new jgl.DList();
        // set the routing of the entity
        sequence.pushBack(new SequenceStepInfo(SimMaster.workstationAtPosition(0),0));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(1),(float)(2 + Math.random())));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(2),(float)(2 + Math.random())));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(3),(float)(2 + Math.random())));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(4),(float)(2 + Math.random())));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(5),(float)(2 + Math.random())));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(6),(float)(2 + Math.random())));
        sequence.pushBack(new
SequenceStepInfo(SimMaster.workstationAtPosition(7),(float)(2 + Math.random())));
        sequence.pushBack(new SequenceStepInfo(SimMaster.workstationAtPosition(8),0));
        i = sequence.begin(); // an iterator at the first item
        currentStep = (SequenceStepInfo)i.get(); // the SequenceStepInfo about this step
        idCounter++;
    }
    // Methods
    public float getProcessingTime() {return currentStep.getProcessingTime();}
    public Workstation getNextWorkstation() {
        i.advance();
        if (i.atEnd()) {
            SimMaster.printEvent("Nothing in Entity Routing List\n");
        }
        // sequence.
        currentStep = (SequenceStepInfo)i.get();
        return currentStep.getWorkstation();
    }
    public int getId() {return id;}
    public static void reset() {idCounter = 1;}
    //Statistical Methods
    public void recordMyStatistics() {stats.recordMyStatistics();}
    // Set-Time Methods
    // Creation
    public void setCreationTime(float t) { stats.setCreationTime(t);}
    // Queue
    public void setQueueEntryTime(float t) {stats.setQueueEntryTime(t);}
    public void setQueueExitTime(float t) {stats.setQueueExitTime(t);}
    // Server
    public void setServerEntryTime(float t) {stats.setServerEntryTime(t);}
    public void setServerExitTime(float t) {stats.setServerExitTime(t);}
}
```

EntityStatistics.java

```
class EntityStatistics {

    // Data fields of this class
    private float creationTime;
    private float terminationTime;
    // Queue-specific Fields
    private float queueEntryTime;
    private float queueExitTime;
    private float specificQueueWaitingTime;
    private float totalAccumulatedQueueWatingTime;
    // Server-specific Fields
    private float serverEntryTime;
    private float serverExitTime;
    private float specificServerProcessingTime;
    // Workstation-specific Fields
    private float specificWorkstationEntryTime;
    private float specificWorkstationExitTime;
    private float specificWorkstationDurationTime;
    // System-specific Fields
    private float timeInSystem;

    EntityStatistics() { //Constructor
        creationTime = SimMaster.getSimTime();
    }
    // Set-Time Methods
    // Creation
    public void setCreationTime(float t) { creationTime = t;}
    // Queue
    public void setQueueEntryTime(float t) {queueEntryTime = t;}
    public void setQueueExitTime(float t) {
        queueExitTime = t;
        calculateTotalAccumulatedQueueWatingTime();
    }
    // Server
    public void setServerEntryTime(float t) {serverEntryTime = t;}
    public void setServerExitTime(float t) {
        serverExitTime = t;
        calculateSpecificServerProcessingTime();
    }
    // Duration Methods
    // Queue
    public void calculateTotalAccumulatedQueueWatingTime() {
        totalAccumulatedQueueWatingTime = queueEntryTime - queueExitTime +
                                          totalAccumulatedQueueWatingTime;
    }
    public void calculateSpecificQueueWaitingTime() {
        specificQueueWaitingTime = queueEntryTime - queueExitTime;
    }
    // Server
    public void calculateSpecificServerProcessingTime() {
        specificServerProcessingTime = serverEntryTime - serverExitTime;
    }
    // Report Stats Upon Exit From the System
    public void recordMyStatistics() {
        timeInSystem = SimMaster.getSimTime() - creationTime;
        // get a ref to the SystemStatistics
        SystemStatistics ss = SimMaster.getSystemStatisticsObject();
        // send as many statis as needed
        ss.recordMyStatistics(timeInSystem);
    }
}
```

RemoteLoader.java

```
class RemoteLoader {
    private static String className;
    private static RemoteWorkstation thisWorkstation;

    public RemoteLoader() {}

    public static void setClassName(String s) {
        className = s;
    }

    public static RemoteWorkstation loadWorkstation() {
        //define the remote workstation
        try {
            thisWorkstation = (RemoteWorkstation)Class.forName(className).newInstance();
        }   catch (ClassNotFoundException e) {
            System.out.println("Caught ClassNotFoundException: " + e.getMessage());
            SimMaster.printEvent("\nCould Not load "+className+".\n\n");
            thisWorkstation = null;
            return thisWorkstation;
        }   catch (InstantiationException e) {
            System.out.println("Caught InstantiationException: " + e.getMessage());
        }   catch (IllegalAccessException e) {
            System.out.println("Caught IllegalAccessException: " + e.getMessage());
        }
        return thisWorkstation;
    }
}
```

RemoteWorkstation.java

```
/**
 * The remote interface is needed to create a Class and an instance of
 * an object that is not know at runtime.  Each remote workstation object must
 * extend the Workstation Class and implement the RemoteWorkstation Interface.
 *@see Workstation
 */
public interface RemoteWorkstation{
    /**
    * Returns the string that identifies a remote workstation.
    * @return The desired String
    */
    public String getWorkstationName();
    /**
    * Triggers the reporting of the workstation's statistics at the end of
    * a run.
    */
    public void reportFinalStatistics();
}
```

SequenceStepInfo.java

```
class SequenceStepInfo {
    // Data Fields
    private Workstation workstation;
    private float processingTime;

    // Constructor
    public SequenceStepInfo(Workstation wks,float pt) {
        workstation = wks;
        processingTime = pt;
    }

    public float getProcessingTime() {return processingTime;}
    public Workstation getWorkstation() {return workstation;}
}
```

SimApplication.java

```
import java.awt.*;

public class SimApplication extends Frame {

    // Data fields of this class
    private SimMaster simMaster = null;
    private String name;
    private String codebase;
    private int eventDelayValue;
    private float runLength;
    private Font font;
    private Font sysfont;

    public SimApplication() {

        font = new Font("Serif",Font.BOLD,12);
        sysfont = new Font("Monospaced",Font.BOLD,12);
        //{{INIT_CONTROLS
        setLayout(null);
        addNotify();
        setSize(getInsets().left + getInsets().right + 588, getInsets().top +
getInsets().bottom + 495);
        eventDisplay=new TextArea(13,54);
        eventDisplay.setFont(sysfont);
        add(eventDisplay);
        eventDisplay.setBounds(getInsets().left + 7,getInsets().top + 208,448,221);
        button1=new Button("Start");
        button1.setFont(font);
        add(button1);
        button1.setBounds(getInsets().left + 5,getInsets().top + 442,91,26);
        button2=new Button("Pause");
        button2.setFont(font);
        add(button2);
        button2.setBounds(getInsets().left + 124,getInsets().top + 442,91,26);
        button3=new Button("Resume");
        button3.setFont(font);
        add(button3);
        button3.setBounds(getInsets().left + 243,getInsets().top + 442,91,26);
        button4=new Button("Quit");
        button4.setFont(font);
        add(button4);
        button4.setBounds(getInsets().left + 481,getInsets().top + 442,91,26);
        Color c = new Color(192,192,192);
        label1=new Label("\"Expose\" Workstation Object:");
        label1.setFont(font);
        label1.setBackground(c);
        add(label1);
        label1.setBounds(getInsets().left + 7,getInsets().top + 13,175,20);
        label2=new Label("Location (URL):");
        label2.setFont(font);
        label2.setBackground(c);
        add(label2);
        label2.setBounds(getInsets().left + 7,getInsets().top + 45,112,20);
        label3=new Label("Current Statistics:");
        label3.setFont(font);
        label3.setBackground(c);
        add(label3);
        label3.setBounds(getInsets().left + 7,getInsets().top + 78,126,13);
        wipDisplay=new TextField(10);
        wipDisplay.setFont(sysfont);
        add(wipDisplay);
        wipDisplay.setBounds(getInsets().left + 7,getInsets().top + 98,84,19);
        cycleTimeDisplay=new TextField(10);
        cycleTimeDisplay.setFont(sysfont);
        add(cycleTimeDisplay);
        cycleTimeDisplay.setBounds(getInsets().left + 7,getInsets().top + 124,84,19);
        throughputDisplay=new TextField(10);
        throughputDisplay.setFont(sysfont);
        add(throughputDisplay);
        throughputDisplay.setBounds(getInsets().left + 7,getInsets().top + 150,84,19);
        label5=new Label("WIP (Units)");
```

```
        label5.setFont(font);
        label5.setBackground(c);
        add(label5);
        label5.setBounds(getInsets().left + 98,getInsets().top + 98,126,19);
        label6=new Label("Cycle Time (Hours)");
        label6.setFont(font);
        label6.setBackground(c);
        add(label6);
        label6.setBounds(getInsets().left + 98,getInsets().top + 124,126,19);
        label7=new Label("Throughput (Units/Hour)");
        label7.setFont(font);
        label7.setBackground(c);
        add(label7);
        label7.setBounds(getInsets().left + 98,getInsets().top + 150,154,19);
        label8=new Label("Current Simulation Time:");
        label8.setFont(font);
        label8.setBackground(c);
        add(label8);
        label8.setBounds(getInsets().left + 336,getInsets().top + 182,140,20);
        simTimeDisplay=new TextField(10);
        simTimeDisplay.setFont(font);
        add(simTimeDisplay);
        simTimeDisplay.setBounds(getInsets().left + 490,getInsets().top + 182,84,20);
        nameField=new TextField(43);
        nameField.setFont(sysfont);
        add(nameField);
        nameField.setBounds(getInsets().left + 203,getInsets().top + 7,357,26);
        codebaseField=new TextField(43);
        codebaseField.setFont(sysfont);
        add(codebaseField);
        codebaseField.setBounds(getInsets().left + 203,getInsets().top + 39,357,26);
        button5=new Button("Reset");
        button5.setFont(font);
        add(button5);
        button5.setBounds(getInsets().left + 362,getInsets().top + 442,91,26);
        check1=new Checkbox("Trace Off");
        check1.setFont(font);
        check1.setBackground(c);
        add(check1);
        check1.setBounds(getInsets().left + 7,getInsets().top + 182,77,19);
        eventDelay=new TextField(7);
        eventDelay.setFont(font);
        add(eventDelay);
        eventDelay.setBounds(getInsets().left + 233,getInsets().top + 182,63,20);
        label4=new Label("Event Delay (ms)");
        label4.setFont(font);
        label4.setBackground(c);
        add(label4);
        label4.setBounds(getInsets().left + 121,getInsets().top + 182,112,13);
        runLengthField=new TextField(10);
        runLengthField.setFont(font);
        add(runLengthField);
        runLengthField.setBounds(getInsets().left + 490,getInsets().top + 91,84,20);
        label9=new Label("Run Length (Hours):");
        label9.setFont(font);
        label9.setBackground(c);
        add(label9);
        label9.setBounds(getInsets().left + 350,getInsets().top + 91,115,20);
        //}}

        // Other inits
        nameField.setText("ExposeS1000");
        codebaseField.setText("ftp://imesupport.ie.orst.edu/NCSOS/");
        eventDelay.setText("500");
        runLengthField.setText("10000");

    }

    public boolean processEvent(Event evt) {
        if (evt.id==Event.WINDOW_DESTROY) System.exit(0);
        return super.handleEvent(evt);
    }

    public boolean action(Event evt, Object arg) {
```

```
        if (arg.equals("Start")) {
            if (simMaster == null) {
                prestart();
                simMaster = new SimMaster(this,runLength);
                simMaster.start();
            }
        }
        if (arg.equals("Pause")) {
            if (simMaster != null) {
                simMaster.suspend();
            }
        }
        if (arg.equals("Resume")) {
            if (simMaster != null) {
                simMaster.resume();
            }
        }
        if (arg.equals("Reset")) {
            if (simMaster != null) {
                reset();
            }
        }
        if (arg.equals("Quit")) {
            System.exit(0);
        }

        else return super.action(evt,arg);
            return true;
    }
    private void prestart() {
        name = nameField.getText();
        codebase = codebaseField.getText();
        runLength = Float.valueOf(runLengthField.getText()).floatValue();
        ClassSaver.saveRemoteClass(name,codebase);
        RemoteLoader.setClassName(name);
//            printEvent("Successfully Loaded: "+name+"\nFrom:"+codebase+"\n");
    }
    public int getDelay() {
        if (check1.getState()) {
            eventDelayValue = 0;
            return eventDelayValue;
        }
        eventDelayValue = Integer.valueOf(eventDelay.getText()).intValue();
        if (eventDelayValue <= 0) { eventDelayValue = 500;}
        return eventDelayValue;
    }
    public static void printSimTime(String s) {
        simTimeDisplay.setText(s);
    }
    public static void updateScreenStatistics(String wip,String tp,String ct) {
        wipDisplay.setText(wip);
        throughputDisplay.setText(tp);
        cycleTimeDisplay.setText(ct);
    }
    public void printEvent(String s) {
        if (!check1.getState()) {
            eventDisplay.append(s);
                if (eventDisplay.getText().length() > 10000) {
                    eventDisplay.replaceRange("",0,(eventDisplay.getText().length()/2));
                }
        }
    }
    public void printError(String s) {
        eventDisplay.append(s);
        simMaster.suspend();
    }

    public static void main(String[] args) {
        Frame f = new SimApplication();
        f.setTitle("NCSOS Simulation Module");
        f.setSize(600,530);
        Color c = new Color(192,192,192);
        f.setBackground(c);
        f.show();
```

```
    }
    private void reset() {
        simMaster.suspend();
        simMaster.reset();
        simMaster.stop();
        simMaster = null;
        eventDisplay.replaceRange("",0,(eventDisplay.getText().length()));
        wipDisplay.setText("0");
        throughputDisplay.setText("0");
        cycleTimeDisplay.setText("0");
        simTimeDisplay.setText("0");
        eventDelay.setText("500");
        check1.setState(false);
    }
    //{{DECLARE_CONTROLS
    static TextArea eventDisplay;
    Button button1;
    Button button2;
    Button button3;
    Button button4;
    Label label1;
    Label label2;
    Label label3;
    static TextField wipDisplay;
    static TextField cycleTimeDisplay;
    static TextField throughputDisplay;
    Label label5;
    Label label6;
    Label label7;
    Label label8;
    static TextField simTimeDisplay;
    static TextField nameField;
    static TextField codebaseField;
    Button button5;
    Checkbox check1;
    TextField eventDelay;
    Label label4;
    TextField runLengthField;
    Label label9;
    //}}

}
```

SimEvent.java

```
import java.lang.reflect.Method;

class SimEvent {  // the base Simulation Event

    // Data fields of this class
    private float time; // delta time the event is int he future
    private Entity eventEntity; //the entity the event is assoc. with
    protected Workstation eventWorkstation;
    protected String eventMethodName;
    protected Method eventMethod;

    //Constructor 1
    public SimEvent(float t, Entity ent, Workstation ws, Method msg) {
        time = t;
        eventEntity = ent;
        eventWorkstation = ws;
        eventMethod = msg;
        eventMethodName = toString(msg);
    }
    //Constructor 2 - no entity
    public SimEvent(float t, Workstation ws, Method msg) {
        time = t;
//        eventEntity = null;
        eventWorkstation = ws;
        eventMethod = msg;
        eventMethodName = toString(msg);
    }

    // Methods
    public Method getEventMethod() {
        return eventMethod;
    }
    public Workstation getEventWorkstation() {
        return eventWorkstation;
    }
    public Entity getEventEntity() {
        return eventEntity;
    }
    public float getEventTime() {
        return time;
    }
    public void setEventTime(float absTime) {
        time = absTime;
    }

    public String getEventMethodName() {
        return eventMethodName;
    }
    public String getEventWorkstationName() {return
eventWorkstation.getWorkstationName();}
    public int getEventEntityId() {
        return eventEntity.getId();
    }
    private String toString(Method msg) {
        String str = msg.toString();
        int s = str.indexOf(".");
        int e = str.indexOf("(");
        str = str.substring(s+1,e);
        return str;
    }
}
```

SimMaster.java

```
import jgl.*;
import java.lang.reflect.Method;
/**
 * The "Simulation Manager." Coordinates and delegates the simulation
 * environment.
 * @author  Kurt Colvin
 */
public class SimMaster extends Thread {  // the manager of the simulation
    // Data fields of this class
    private static SimApplication ui; // ref back to the SimApplication
    private static float simTime; // the simulation clock
    private static jgl.DList fel; //the future events lists
    private static SimModel fab; // the semiconductor fab. model
    private static Creator entityCreator;
    private static int delay = 100;//delay between events
    private static SystemStatistics systemStatistics;
    private float runLength; //length of simulation run
    /**
    * Initializes the thread, creates a reference back to the
    * User Interface, creates the Future Events List, creates the
    * simulation model, creates the entity creator, and sets the SimTime
    * to zero.
    * @param sa The reference back to the SimApplication (User Interface).
    */
    public SimMaster(SimApplication sa, float rl) {  // Constructor
        super(); // init the thread
        ui = sa;
        fel = new jgl.DList(); // create the fel
        fab = new SimModel(); // create the model
        systemStatistics = new SystemStatistics(); // create the SystemStats
        entityCreator = new Creator(this); // The entity creator for the whole model
        simTime = 0; //set simTime to zero
        runLength = rl;
    }
    /**
    * The main loop for the simulation run.  Will execute until
    * the run length has been reached, or there are no more events
    * in the future events list
    */
    public void run() {  //Run the Thread
        ui.printEvent("Push the \"Resume\" Button to continue.\n");
        this.suspend();
        printEvent("Starting Simulation:");
        for(int i=1;i<20;i++) {
            newEntityIn((float)0);// start sim with a few entities
        }
//        while(!fel.isEmpty()) {
        while(simTime < runLength) {
            delay = ui.getDelay();
            if (fel.isEmpty()) {
                printEvent(new String("Nothing in fel \n"));
            }
            SimEvent nextEvent = (SimEvent)fel.popFront(); //get the next event
            printEvent(nextEvent);
            simTime = nextEvent.getEventTime();  // set the simTime
            SimApplication.printSimTime(String.valueOf(simTime)); // print the simTime to
the appliation
            Workstation eventWorkstation = (Workstation)nextEvent.getEventWorkstation();
// get the WS
            Object[] args = new Object[1];
            args[0] = nextEvent.getEventEntity();
            if (args[0] == null) {args[0] = new Object();}
            Method eventMethod = null;
            try { eventMethod = nextEvent.getEventMethod();}
            catch (NullPointerException e) {
                System.out.println("From SimMaster: Caught NullPointerExcpetion:
"+e.getMessage());
            }
//            System.out.println("From SimMaster:Calling:
"+eventMethod+"("+eventWorkstation+","+args[0]+")");
            try {
```

```
                eventMethod.invoke(eventWorkstation,args);
            }  catch (NullPointerException e) {
                System.out.println("from invoke"+e.getMessage());
            }  catch (IllegalAccessException e) {
                System.out.println("From SimMaster: Caught IllegalAccessException:
"+e.getMessage());
            }  catch (IllegalArgumentException e) {
                System.out.println("From SimMaster: Caught IllegalArgumentException:
"+e.getMessage());
            }  catch (java.lang.reflect.InvocationTargetException e) {
                System.out.println("From SimMaster: Caught InvocationTargetException:
"+e.getMessage());
            }
//            System.out.println("Return from invoke");
            try {
//                Thread.yield();
                Thread.sleep(delay);
            }
            catch (java.lang.InterruptedException e) {}
        }
        ui.printEvent("\nEnd of Simulation");
        fab.reportFinalStatistics();
    }
    /**
    * Schedules the events in the correct order into the future event list.
    * @param newEvent The new event to be scheduled.
    */
    public static void scheduleEvent(SimEvent newEvent) { //insert the event into fel
        newEvent.setEventTime(simTime + newEvent.getEventTime()) ;//convert to abs time
        // if fel is empty, just add the event to the front
        if (fel.isEmpty()) {
            fel.pushBack(newEvent);
            return;
        }
        // Do some error checking to make sure we don't schedule an event in the past
        SimEvent firstEvent = (SimEvent)fel.front();
        if (newEvent.getEventTime() < simTime) {
            String error = new String("Can't schedule an Event in the Past! \n");
            SimMaster.printError(error);
            return;
        }

        // iterate through the fel until the time of the newEvent is less than or equal to
        // the currentEvent.  When this is true, we're in the right spot, so insert the
        // event into that location.
        DListIterator i = fel.begin();
        while (i.hasMoreElements()) {
            SimEvent currentEvent = (SimEvent)i.get();
            if (newEvent.getEventTime() <= currentEvent.getEventTime()) {
                fel.insert(i,newEvent); // insert here.
                return;
            }
            i.advance(); // otherwise advance iterator.
        }
        // if you get here, EventTime is more than all others, so add to end
        fel.pushBack(newEvent);
    }
    /**
    * Asks the SystemStatistics object to return the SystemStatistics object
    */
    public static SystemStatistics getSystemStatisticsObject() {return systemStatistics;}
    /**
    * Asks the SimModel object to return a workstation at a certain location.
    */
    public static Workstation workstationAtPosition(int n) {
        return fab.workStationAtPosition(n);
    }
    /**
    * Returns the current Simulation Time.
    */
    public static float getSimTime() {return simTime;}
    /**
    * Creates a String from an event, and passes it to the User Interface
    * @param evt The SimEvent to be printed.
```

```
    */
    public static void printEvent(SimEvent evt) { // time, entity,method, workstation
        int id;
        try { id = evt.getEventEntityId();
        }  catch (NullPointerException e) {
            id = 0;
        }
        if (id == 0) {
            ui.printEvent(""+evt.getEventTime()+"   "+evt.getEventMethodName()+" on "
                          +evt.getEventWorkstationName()+"\n");
        } else {
            ui.printEvent(""+evt.getEventTime()+" Entity: "+evt.getEventEntityId()+
                    " "+evt.getEventMethodName()+" on
"+evt.getEventWorkstationName()+"\n");
        }
    }
    /**
    * Just passes a string onto the User Interface
    *@param s The String to be forwarded.
    */
    public static void printEvent(String s) {ui.printEvent("\t"+s+"\n");}
    /**
    * Passes a String onto the User Interface.  The execution will be halted as a result
    * of some error occurring.
    *@param error The string to be forwarded.
    */
    public static void printError(String error) {ui.printError(error);}
    /**
    * Passes on 3 strings to the User Interface.
    *@param wip The WIP value.
    *@param tp The Throughput value.
    *@param ct The Cycle Time Value.
    */
    public static void updateScreenStatistics(String wip,String tp,String ct) {
        ui.updateScreenStatistics(wip,tp,ct);
    }
    /**
    * Passes a request for a newly created Entity to the entityCreator object.
    *@param dt The delay before the new entity will be scheduled for arrival to the
system.
    */
    public static void newEntityIn(float dt) {entityCreator.newEntityIn(dt);}
    public static void newEntity() {entityCreator.newEntity();}
    /**
    * Passes a request to the SimModel object to get a reference to a workstation
    * at a certain location.
    *@return The workstation at the index.
    */
    public Workstation getArrivalLocation(){
        return fab.getArrivalLocation();
    }
    /**
    * Sends the updateWIP request to the SystemStatistics Object.
    */
    public static void updateWIP(int i) {
        systemStatistics.updateWIP(i);
    }
    public float getWIP() {
        return systemStatistics.getWIP();
    }
    /**
    * Resets the application so a new run can be performed.
    */
    public static void reset() { //used to clear all variables
        Entity.reset();
        simTime = 0;
        fab = null;
        fel = null;
        entityCreator = null;
    }
}
```

SimModel.java

```
import jgl.DList;
class SimModel {  // the Model representing the Fab

    // Data fields of this class
    private ArrivalLocation arrival;
    private ApplyWorkstation workstation1;
    private RemoteWorkstation workstation2;
    private DevelopWorkstation workstation3;
    private BakeWorkstation workstation4;
    private DepositWorkstation workstation5;
    private EtchWorkstation workstation6;
    private TestWorkstation workstation7;
    private TerminationLocation terminate;

    private SystemStatistics systemStatistics;
    private DList workstationLayout;

    SimModel() {   // Constructor
        //needed objects
        workstationLayout = new jgl.DList(); // the list of Workstation Objects, in order
        // add arrival Location
        arrival= new ArrivalLocation(); // interarrival time
        // define local workstations, must be in order.
        workstation1 = new ApplyWorkstation();
        //define the remote workstation
        workstation2 = (RemoteWorkstation)RemoteLoader.loadWorkstation();
        if (workstation2 == null) {
            SimMaster.printEvent("Did NOT Successfully Load Workstation\n");
            SimMaster.printEvent("Please push \"Reset\" and enter a valid Name and
URL\n");
        } else {SimMaster.printEvent("Successfully Loaded Workstation\n");}
        //then the rest of the local workstations
        workstation3 = new DevelopWorkstation();
        workstation4 = new BakeWorkstation();
        workstation5 = new DepositWorkstation();
        workstation6 = new EtchWorkstation();
        workstation7 = new TestWorkstation();

        // Finally Add Terminate Location
        terminate = new TerminationLocation();
        // Build workstationLayout
        workstationLayout.pushBack(arrival);//put ws in list
        workstationLayout.pushBack(workstation1);//put ws in list
        workstationLayout.pushBack(workstation2);//put ws in list
        workstationLayout.pushBack(workstation3);//put ws in list
        workstationLayout.pushBack(workstation4);//put ws in list
        workstationLayout.pushBack(workstation5);//put ws in list
        workstationLayout.pushBack(workstation6);//put ws in list
        workstationLayout.pushBack(workstation7);//put ws in list

        workstationLayout.pushBack(terminate);//put ws in list
    }
    public void scheduleEvent(SimEvent evt) {
        SimMaster.scheduleEvent(evt);
    }
    // Workstation methods
    public Workstation getArrivalLocation() {return arrival;}
    public Workstation workStationAtPosition(int n) {
        return (Workstation)workstationLayout.at(n);
    }
    // Final Statistics
    public void reportFinalStatistics() {
        workstation1.reportFinalStatistics();
        workstation2.reportFinalStatistics();
        workstation3.reportFinalStatistics();
        workstation4.reportFinalStatistics();
        workstation5.reportFinalStatistics();
        workstation6.reportFinalStatistics();
        workstation7.reportFinalStatistics();}}
```

SystemStatistics.java

```
class SystemStatistics {  // keeps track of system stats

    // Observation Based Statictics
    // Time in System/Cycle Time vaiables
    private float averageTimeInSystem = 0;
    private float cumulativeTimeInSystem = 0;
    private int numberOfEntitiesExited = 0;
    private float sumOfTimeInSystemDeviationsSquared = 0;
    private float timeInSystemMin = 0;
    private float timeInSystemMax = 0;
    // Throughput Variables
    private float averageThroughput = 0;
    private float currentThroughput = 0;
    private float cumulativeThroughput = 0;
    private float sumOfThroughputDeviationsSquared = 0;
    private float throughputMin = 0;
    private float throughputMax = 0;
    // Time-Based Statistics
    // WIP
    private float currentWIP = 0;
    private float tLast = 0; // Time of Last observation
    private float averageWIP = 0;
    private float cumulativeWIP = 0;
    private float sumOfWIPDeviationsSquared = 0;
    private float WIPMax = 0;
    private float WIPMin = 0;
    // Other Variables
    SystemStatistics() {   // Constructor
    }

    // Methods
    public void updateStatistics() {
        String wp = new String(""+currentWIP+"");
        String tp = new String(""+averageThroughput+"");
        String ct = new String(""+averageTimeInSystem+"");
        SimMaster.updateScreenStatistics(wp,tp,ct);
    }
    public void recordMyStatistics(float tis) {
        numberOfEntitiesExited++;
        // Cycle Time
        cumulativeTimeInSystem = cumulativeTimeInSystem + tis;
        averageTimeInSystem = cumulativeTimeInSystem/numberOfEntitiesExited;
            double tisdiff = averageTimeInSystem - tis;
        sumOfTimeInSystemDeviationsSquared =  (float)Math.pow(tisdiff,2) +
                                              sumOfTimeInSystemDeviationsSquared;
        timeInSystemMin = Math.min(tis,timeInSystemMin);
        timeInSystemMax = Math.max(tis,timeInSystemMax);
        // Throughput
        averageThroughput = numberOfEntitiesExited/SimMaster.getSimTime();
/*        currentThroughput = numberOfEntitiesExited/SimMaster.getSimTime();
        cumulativeThroughput = currentThroughput + cumulativeThroughput;
        averageThroughput = cumulativeThroughput/numberOfEntitiesExited;
            double tpdiff = averageThroughput - currentThroughput;
        sumOfThroughputDeviationsSquared = (float)Math.pow(tpdiff,2) +
                                           sumOfThroughputDeviationsSquared;
        throughputMin = Math.min(currentThroughput,throughputMin);
        throughputMax = Math.max(currentThroughput,throughputMax);
*/        // Update the screen
        updateStatistics();
    }
    //Time-Based WIP
    public void updateWIP(int n) { // adds or subtracts WIP
        currentWIP = currentWIP + n;
        float dt = SimMaster.getSimTime() - tLast;
        cumulativeWIP = (currentWIP * dt) + cumulativeWIP;
        averageWIP = cumulativeWIP/SimMaster.getSimTime();
            double WIPdiff = averageWIP - currentWIP;
        sumOfWIPDeviationsSquared = (float)Math.pow(WIPdiff,2) +
                                    sumOfWIPDeviationsSquared;
        WIPMin = Math.min(currentWIP,WIPMin);
        WIPMax = Math.max(currentWIP,WIPMax);
```

```
        tLast = SimMaster.getSimTime();
        updateStatistics();
    }
    public float getWIP() {
        return currentWIP;
    }
}
```

TerminationLocation.java

```
class TerminationLocation extends Workstation {  // Exit Point

    // Data fields of this class
    private float dt = 0; //used to schedule events
    private int servers; // number of servers in this workstation

    //Constructor
    TerminationLocation() {
        super();
        workstationName = new String("Termination Location");
    }
    // Methods
    public void arrive(Entity ent) {
        // create the next entity
        SimMaster.newEntity(); // create next entity
        // Collect stats
        SimMaster.printEvent("Entity "+ent.getId()+" Collecting Statistics");
        ent.recordMyStatistics();
        SimMaster.updateWIP(-1);
        super.arrive(ent);
    }
    public void depart(Entity ent) {
        //Entity exits the System
        SimMaster.printEvent("Entity "+ent.getId()+" Exited the System");
    }
}
```

Workstation.java

```
import java.awt.*;
import java.lang.reflect.Method;
/**
 * The Abstract Workstation Class.
 * Defines the basic behavior of a workstation.
 *
 * @author  Kurt Colvin
 * @see RemoteWorkstation
 */
public class Workstation {
    /**
    * The String that describes the workstation. 
    * Assigned when an instance of a subclass is created.
    */
    protected String workstationName; // name of workstation
    /**
    * An array of Class Objects. Used for internal method calls
    */
    protected Class[] args; // an array of classes
    /**
    * An array of Class Objects. Used for internal method calls
    */
    protected Class[] nullargs; // an array of classes
    /**
    * A java.lang.reflect.Method Object.  Used for internal method calls
    * @see java.lang.reflect.Method
    */
    protected Method msg;
    /**
    * A java.awt.TextArea to write statistics to
    *
    * @see java.awt.TextArea
    */
    private TextArea out; //stats output
    /**
    * The constructor for the Workstation class.
    * Gets called when a subclass is created.
    * Basically sets up the internal data fields for
    * method processing.
    */
    public Workstation() {   // Constructor
        // java.lang.reflect related stuff
        args = new Class[1];
        nullargs = new Class[1];
        try {
            args[0] = Class.forName("Entity");
            nullargs[0] = Class.forName("java.lang.Object");
        }  catch (ClassNotFoundException e) {}
        workstationName = null; // will be assigned in each subclass
    }
    /**
    * An arrive method that will be called if the subclass
    * does not define its own "arrive" method.  This method
    * schedules an "depart" event in 0 time.
    * @param ent the Entity that is associated with the "arrive event."
    */
    public void arrive(Entity ent) {
        float dt = 0;
        try {
           msg = this.getClass().getMethod("depart",args);
        }  catch (NoSuchMethodException e) {
            System.out.println("From Workstation:arrive() NoSuchMethodException:
"+e.getMessage());
        }
        SimMaster.scheduleEvent(new SimEvent(dt,ent,this,msg));
    }
    /**
    * A depart method that will be called if the subclass
    * does not define its own "depart" method.  This method
    * schedules an "arrive" event on the next workstation in in 0 time.
    *
```

```
    * @param ent the Entity that is associated with the "arrive event."
    */
    public void depart(Entity ent) {
        float dt = 0;
        Workstation nextWorkstation = ent.getNextWorkstation();
        try {
           msg = nextWorkstation.getClass().getMethod("arrive",args);
        }  catch (NoSuchMethodException e) {
            System.out.println("From Workstation: depart() NoSuchMethodException:
"+e.getMessage());
        }
        SimMaster.scheduleEvent(new SimEvent(dt,ent,nextWorkstation,msg));
    }
    /**
    * Attempts to schedule the event on the future events list.  The subclasses
    * will pass the parameters for an event to this method, and this method
    * will create a new event (from the parameters) and pass the event to the
    * SimMaster object.
    *
    * @param dt The duration of time the event will be scheduled in the future
    * @param ent The Entity associated with the event.
    * @param method The String name of the type of event to be scheduled.
    */
    protected void scheduleEvent(float dt, Entity ent, String method) {
        try {
           msg = this.getClass().getMethod(method,args);
        }  catch (NoSuchMethodException e) {
            System.out.println("From Workstation: scheduleEvent1() NoSuchMethodException:
"+e.getMessage());
        }
        SimMaster.scheduleEvent(new SimEvent(dt,ent,this,msg));
    }
    // schedule an event that is not associated with an entity
    /**
    * Attempts to schedule the event on the future events list.  The subclasses
    * will pass the parameters for an event to this method, and this method
    * will create a new event (from the parameters) and pass the event to the
    * SimMaster object.
    *
    * @param dt The duration of time the event will be scheduled in the future
    * @param method The String name of the type of event to be scheduled.
    */
    protected void scheduleEvent(float dt,String method) {
        try {
           msg = this.getClass().getMethod(method,nullargs);
        }  catch (NoSuchMethodException e) {
            System.out.println("From Workstation: scheduleEvent2() NoSuchMethodException:
"+e.getMessage());
        }
        SimMaster.scheduleEvent(new SimEvent(dt,this,msg));
    }
    /**
    * A simple method to return the String that describes the Workstation.
    *
    * @return The desired String.
    */
    public String getWorkstationName() {return workstationName;}
    /**
    * Prints a string to the Statistics Window
    *
    * @param s The String to print.
    */
    protected void printStat(String s) {
        out.append(s+"\n");
    }
    /**
    * Creates a Frame to output statistics.
    *
    * @param s The Name of the Frame.
    */
    protected void makeStatisticsWindow(String s) {
        Frame f = new Frame(s);
        f.setLocation(400,200);
        f.setSize(500,200);
```

```
        f.setResizable(false);
        out = new TextArea(5,54);
        f.add(out);
        out.setBounds(f.getInsets().left,f.getInsets().top ,490,190);
        f.show();
    }
}
```

APPENDIX B: Java Local Workstation Class Definitions

ApplyWorkstation.java

```
import jgl.DList;

class ApplyWorkstation extends Workstation {  // First workstation

    // Data fields
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a FIFO queue
    float inUse; //status of server
    float startService; //start of service
    //Constructor
    ApplyWorkstation() {
        super();
        workstationName = new String("Apply Workstation");
        servers = 1;
        queue = new DList();
        inUse = 0;
        startService = 0;
    }
    public void arrive(Entity ent) {
        queue.pushBack(ent); // put entity into queue
        super.scheduleEvent(0,"load");
    }
    public void load(Object obj) { // dummy object
        if (servers == 0) {
            SimMaster.printEvent("Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Queue.");
            return;
        }
        // Otherwise, load the entity
        servers--;
        startService = SimMaster.getSimTime();
        Entity ent = (Entity)queue.popFront(); // get the first entity in queue
        float dt = ent.getProcessingTime();
        super.scheduleEvent(dt,ent,"unload");
    }
    public void unload(Entity ent) {
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        super.scheduleEvent(0,"load"); // check the queue
        super.scheduleEvent(0,ent,"depart");
    }
    public void reportFinalStatistics() {
        super.makeStatisticsWindow("Apply Workstation Final Statistics");
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

BakeWorkstation.java

```
import jgl.DList;

class BakeWorkstation extends Workstation {  // First workstation

    // Data fields
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a FIFO queue
    float inUse; //status of server
    float startService; //start of service
    //Constructor
    BakeWorkstation() {
        super();
        workstationName = new String("Bake Workstation");
        servers = 1;
        queue = new DList();
        inUse = 0;
        startService = 0;
    }
    public void arrive(Entity ent) {
        queue.pushBack(ent); // put entity into queue
        super.scheduleEvent(0,"load");
    }
    public void load(Object obj) { // dummy object
        if (servers == 0) {
            SimMaster.printEvent("Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Queue.");
            return;
        }
        // Otherwise, load the entity
        servers--;
        startService = SimMaster.getSimTime();
        Entity ent = (Entity)queue.popFront(); // get the first entity in queue
        float dt = ent.getProcessingTime();
        super.scheduleEvent(dt,ent,"unload");
    }
    public void unload(Entity ent) {
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        super.scheduleEvent(0,"load"); // check the queue
        super.scheduleEvent(0,ent,"depart");
    }
    public void reportFinalStatistics() {
        super.makeStatisticsWindow("Bake Workstation Final Statistics");
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

DepositWorkstation.java

```
import jgl.DList;

class DepositWorkstation extends Workstation {  // First workstation

    // Data fields
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a FIFO queue
    float inUse; //status of server
    float startService; //start of service
    //Constructor
    DepositWorkstation() {
        super();
        workstationName = new String("Deposit Workstation");
        servers = 1;
        queue = new DList();
        inUse = 0;
        startService = 0;
    }
    public void arrive(Entity ent) {
        queue.pushBack(ent); // put entity into queue
        super.scheduleEvent(0,"load");
    }
    public void load(Object obj) { // dummy object
        if (servers == 0) {
            SimMaster.printEvent("Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Queue.");
            return;
        }
        // Otherwise, load the entity
        servers--;
        startService = SimMaster.getSimTime();
        Entity ent = (Entity)queue.popFront(); // get the first entity in queue
        float dt = ent.getProcessingTime();
        super.scheduleEvent(dt,ent,"unload");
    }
    public void unload(Entity ent) {
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        super.scheduleEvent(0,"load"); // check the queue
        super.scheduleEvent(0,ent,"depart");
    }
    public void reportFinalStatistics() {
        super.makeStatisticsWindow("Deposit Workstation Final Statistics");
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

DevelopWorkstation.java

```
import jgl.DList;

class DevelopWorkstation extends Workstation {

    // Data fields of this class
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a FIFO queue
    float inUse; //status of server
    float startService; //start of service
    //Constructor
    DevelopWorkstation() {
        super();
        workstationName = new String("Develop Workstation");
        servers = 1;
        queue = new DList();
        inUse = 0;
        startService = 0;
    }
    public void arrive(Entity ent) {
        queue.pushBack(ent); // put entity into queue
        super.scheduleEvent(0,"load");
    }
    public void load(Object obj) { // dummy object
        if (servers == 0) {
            SimMaster.printEvent("Develop Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Develop Queue.");
            return;
        }
        // Otherwise, load the entity
        servers--;
        startService = SimMaster.getSimTime();
        Entity ent = (Entity)queue.popFront(); // get the first entity in queue
        float dt = ent.getProcessingTime();
        super.scheduleEvent(dt,ent,"unload");
    }
    public void unload(Entity ent) {
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        super.scheduleEvent(0,"load"); // check the queue
        super.scheduleEvent(0,ent,"depart");
    }
    public void reportFinalStatistics() {
        super.makeStatisticsWindow(workstationName+" Final Statistics");
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

EtchWorkstation.java

```
import jgl.DList;

class EtchWorkstation extends Workstation {  // First workstation

    // Data fields
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a FIFO queue
    float inUse; //status of server
    float startService; //start of service
    //Constructor
    EtchWorkstation() {
        super();
        workstationName = new String("Etch Workstation");
        servers = 1;
        queue = new DList();
        inUse = 0;
        startService = 0;
    }
    public void arrive(Entity ent) {
        queue.pushBack(ent); // put entity into queue
        super.scheduleEvent(0,"load");
    }
    public void load(Object obj) { // dummy object
        if (servers == 0) {
            SimMaster.printEvent("Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Queue.");
            return;
        }
        // Otherwise, load the entity
        servers--;
        startService = SimMaster.getSimTime();
        Entity ent = (Entity)queue.popFront(); // get the first entity in queue
        float dt = ent.getProcessingTime();
        super.scheduleEvent(dt,ent,"unload");
    }
    public void unload(Entity ent) {
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        super.scheduleEvent(0,"load"); // check the queue
        super.scheduleEvent(0,ent,"depart");
    }
    public void reportFinalStatistics() {
        super.makeStatisticsWindow("Etch Workstation Final Statistics");
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

TestWorkstation.java

```
import jgl.DList;

class TestWorkstation extends Workstation {  // First workstation

    // Data fields
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a FIFO queue
    float inUse; //status of server
    float startService; //start of service
    //Constructor
    TestWorkstation() {
        super();
        workstationName = new String("Test Workstation");
        servers = 1;
        queue = new DList();
        inUse = 0;
        startService = 0;
    }
    public void arrive(Entity ent) {
        queue.pushBack(ent); // put entity into queue
        super.scheduleEvent(0,"load");
    }
    public void load(Object obj) { // dummy object
        if (servers == 0) {
            SimMaster.printEvent("Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Queue.");
            return;
        }
        // Otherwise, load the entity
        servers--;
        startService = SimMaster.getSimTime();
        Entity ent = (Entity)queue.popFront(); // get the first entity in queue
        float dt = ent.getProcessingTime();
        super.scheduleEvent(dt,ent,"unload");
    }
    public void unload(Entity ent) {
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        super.scheduleEvent(0,"load"); // check the queue
        super.scheduleEvent(0,ent,"depart");
    }
    public void reportFinalStatistics() {
        super.makeStatisticsWindow("Test Workstation Final Statistics");
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

APPENDIX C: Java ExposeWorkstation Class Definition

ExposeWorkstation.java

```
import jgl.DList; // a linked-list from the Java Generic Library

// the name of the class is upto the analyst.
// it is mandatory to extend Workstation and implement RemoteWorkstation
class ExposeWorkstation extends Workstation implements RemoteWorkstation {

    // Data fields of this class
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a double linked-list from the jgl package
                         // could use somethink like an Array also.
    float inUse; //status of server
    float startService; //start of service

    //Constructor
    ExposeWorkstation() {
        // init the Workstation object
        super();
        // give this a name
        workstationName = "Remote Expose Workstation";
        // number of servers in this workstation
        servers = 1;
        // create a queue to store entity waiting for processing
        queue = new DList();
        // status variable
        inUse = 0;
        // time marker
        startService = 0;
    }
    // Overrides the arrive method in the Workstation class.
    //Needed if any processing is going to be done at this workstation.
    public void arrive(Entity ent) {
        // put entity into queue
        queue.pushBack(ent);
        // schedule a load event in "0" time delay
        super.scheduleEvent(0,"load");
    }
    // Define a "load" event.
    // the parameter Object is a dummy.  must be used if the event
    // is not Entity-related.
    public void load(Object obj) {
        if (servers == 0) {
            SimMaster.printEvent("Expose Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in Expose Queue.");
            return;
        }
        // Otherwise, load the entity
        // decrement the servers avaiable.
        servers--;
        // mark start time
        startService = SimMaster.getSimTime();
        // get the first entity in queue
        Entity ent = (Entity)queue.popFront();
        SimMaster.printEvent("Successfully loaded Entity: #"+ent.getId());
        // get the processing time from the entity
        float dt = ent.getProcessingTime();
        // schedule the unload event
        super.scheduleEvent(dt,ent,"unload");
    }
    // define the "unload" event
    // notice this is an "entity-related" method.
```

```
    // the related entity is the parameter
    public void unload(Entity ent) {
        // increment the servers available.
        servers++;
        inUse = inUse + (SimMaster.getSimTime() - startService);
        // check the queue, non-entity related
        super.scheduleEvent(0,"load");
        //schedule a depart event immediately, entity related
        super.scheduleEvent(0,ent,"depart");
    }

    // No need to define the "depart" event here.  Just let the super object
    // take care of sending the entity to the next workstation.

    public void reportFinalStatistics() { // called at the end of the run
        super.makeStatisticsWindow(workstationName+" Final Statistics");// (mandatory
line)
        //report the collected stats in the following lines.
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

APPENDIX D: Java ExposeS1000 Class Definition

ExposeS1000.java

```
import jgl.DList; // a linked-list from the Java Generic Library

// the name of the class is upto the analyst.
// it is mandatory to extend Workstation and implement RemoteWorkstation
class ExposeS1000 extends Workstation implements RemoteWorkstation {

    // Data fields of this class
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a double linked-list from the jgl package
                         // could use somethink like an Array also.
    private float inUse; //utilization stat
    private boolean busy; //status variable
    private float startService; //start of service
    private Entity wafer; // wafer that the server is currently working on

    //Constructor
    ExposeS1000() {
        // init the Workstation object
        super();
        // give this a name
        workstationName = "Remote Expose S1000";
        // number of servers in this workstation
        servers = 1;
        // create a queue to store entity waiting for processing
        queue = new DList();
        // util stat
        inUse = 0;
        // status variable
        busy = false;
        // time marker
        startService = 0;
        // wafer that the server is currently working on
        wafer = null;
        // schedule first breakdown
        float dt = (float) (100 * Math.random());
        super.scheduleEvent(dt,"failure");

    }
    // Overrides the arrive method in the Workstation class.
    //Needed if any processing is going to be done at this workstation.
    public void arrive(Entity ent) {
        // put entity into queue
        queue.pushBack(ent);
        // schedule a load event in "0" time delay
        super.scheduleEvent(0,"tryToLoad");
    }
    // Define a "TryToLoad" event.
    // the parameter Object is a dummy.  must be used if the event
    // is not Entity-related.
    public void tryToLoad(Object obj) {
        if (servers == 0) {
            SimMaster.printEvent("S1000 Server Not Available: Put into Queue.");
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in S1000 Queue.");
            return;
        }
        // Otherwise, schedule a "load"
        // decrement the servers avaiable.
        servers--;
        busy = true;
        // mark start time
```

```
        startService = SimMaster.getSimTime();
        // get the first entity in queue
        Entity ent = (Entity)queue.popFront();
        wafer = ent; // set the wafer currently being served
        SimMaster.printEvent("Will process Wafer: # "+ent.getId());
        // load time is uniform(.16,.33) hours
        float dt = (float)(.16 + .16 * Math.random());
        // schedule the load event
        super.scheduleEvent(dt,ent,"load");
    }
    // define the load event
    public void load(Entity ent) {
        SimMaster.printEvent("S1000 Loaded Wafer: # "+ent.getId());
        // get the processing time from the entity
        float dt = ent.getProcessingTime();
        // schedule the unload event
        super.scheduleEvent(dt,ent,"unload");
    }
    // define the "unload" event
    // notice this is an "entity-related" method.
    // the related entity is the parameter
    public void unload(Entity ent) {
        SimMaster.printEvent("S1000 UnLoaded Wafer: # "+ent.getId());
        // increment the servers available.
        servers++;
        busy = false;
        wafer = null; //set the current wafer back to null
        inUse = inUse + (SimMaster.getSimTime() - startService);
        // check the queue, non-entity related
        super.scheduleEvent(0,"tryToLoad");
        // unload time is uniform(.16,.33) hours
        float dt = (float)(.16 + .16 * Math.random());
        super.scheduleEvent(dt,ent,"depart");
    }

    // No need to define the "depart" event here.  Just let the super object
    // take care of sending the entity to the next workstation.

    // unscheduled break down
    public void failure(Object obj) {
        SimMaster.printEvent("Server Failure Ocurred on S1000");
//        SimMaster.printError("Push \"Resume\" to continue");
        // current entity get destroyed
        if (busy) {
//            queue.pushFront(wafer);
            busy = false;
            SimMaster.printEvent("Wafer: # "+wafer.getId()+"was destroyed");
            inUse = inUse + (SimMaster.getSimTime() - startService);
        } else {
            SimMaster.printEvent("Server was idle");
        }
        // take the server out of service
        servers = 0;
        // schedule the returnToService time
        float dt = (float)(1 + Math.random());
//        System.out.println(dt);
        super.scheduleEvent(dt,"returnToService");
    }
    // fixed and returned to service
    public void returnToService(Object obj) {
        SimMaster.printEvent("Server returned to service on S1000");
//        SimMaster.printError("Push \"Resume\" to continue");
        servers = 1;
        super.scheduleEvent(0,"tryToLoad");
        // schedule next failure
        float dt = (float) (100 * Math.random());
//        System.out.println(dt);
        super.scheduleEvent(dt,"failure");
    }
    public void reportFinalStatistics() { // called at the end of the run
        super.makeStatisticsWindow(workstationName+" Final Statistics");// (mandatory
line)
        //report the collected stats in the following lines.
        super.printStat("The server Utilization was: "+inUse/SimMaster.getSimTime());
```

```
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

APPENDIX E: Java ExposeS2000 Class Definition

ExposeS2000.java

```
import jgl.DList; // a linked-list from the Java Generic Library

// the name of the class is upto the analyst.
// it is mandatory to extend Workstation and implement RemoteWorkstation
class ExposeS2000 extends Workstation implements RemoteWorkstation {

    // Data fields of this class
    private float processingTime; //processing delay
    private int servers; // number of servers in this workstation
    private DList queue; // a double linked-list from the jgl package
                         // could use somethink like an Array also.
    private float inUse1; //utilization stat
    private boolean busy1; //status variable
    private float inUse2; //utilization stat
    private boolean busy2; //status variable

    private float startService1; //start of service
    private Entity wafer1; // wafer that the server is currently working on
    private float startService2; //start of service
    private Entity wafer2; // wafer that the server is currently working on

    //Constructor
    ExposeS2000() {
        // init the Workstation object
        super();
        // give this a name
        workstationName = "Remote Expose S2000";
        // number of servers in this workstation
        servers = 2;
        // create a queue to store entity waiting for processing
        queue = new DList();
        // util stat
        inUse1 = 0;
        inUse2 = 0;
        // status variable
        busy1 = false;
        busy2 = false;
        // time marker
        startService1 = 0;
        startService2 = 0;
        // wafer that the server is currently working on
        wafer1 = null;
        wafer2 = null;
        // schedule first breakdown
        float dt = (float) (60 * Math.random());
        super.scheduleEvent(dt,"failure1");
        dt = (float) (60 * Math.random());
        super.scheduleEvent(dt,"failure2");

    }
    // Overrides the arrive method in the Workstation class.
    //Needed if any processing is going to be done at this workstation.
    public void arrive(Entity ent) {
        // put entity into queue
        queue.pushBack(ent);
        // schedule a load event in "0" time delay
        super.scheduleEvent(0,"tryToLoad");
    }
    // Define a "TryToLoad" event.
    // the parameter Object is a dummy.  must be used if the event
    // is not Entity-related.
    public void tryToLoad(Object obj) {
        if (servers == 0) {
            SimMaster.printEvent("S2000 Server Not Available: Put into Queue.");
```

```
            return;
        }
        if (queue.isEmpty()) {
            SimMaster.printEvent("Nothing in S2000 Queue.");
            return;
        }
        // Otherwise, schedule a "load"
        // decrement the servers avaiable.
        servers--;
        // get the entity
        Entity ent = (Entity)queue.popFront();
        // set the busy boolean
        if(busy1) { // going to use server 2
            busy2=true;
            startService2 = SimMaster.getSimTime();
            wafer2 = ent; // set the wafer currently being served
            SimMaster.printEvent("Will process Wafer: # "+ent.getId()+" on Server 2");
        }
        else { //going to usre server 1
            busy1=true;
            startService1 = SimMaster.getSimTime();
            wafer1 = ent; // set the wafer currently being served
            SimMaster.printEvent("Will process Wafer: # "+ent.getId()+" on Server 1");
        }
        // load time is uniform(.5,.66) hours
        float dt = (float)(.5 + .16 * Math.random());
        // schedule the load event
        super.scheduleEvent(dt,ent,"load");
    }
    // define the load event
    public void load(Entity ent) {
        SimMaster.printEvent("S2000 Loaded Wafer: # "+ent.getId());
        // get the processing time from the entity
        float dt = ent.getProcessingTime();
        // schedule the unload event
        super.scheduleEvent(dt,ent,"unload");
    }
    // define the "unload" event
    public void unload(Entity ent) {
        // figure out which server this event is on
        if(ent == wafer1){ //then we are working with server 1
            SimMaster.printEvent("S2000 UnLoaded Wafer: # "+ent.getId()+" from Server 1");
            // increment the servers available.
            servers++;
            busy1 = false;
            wafer1 = null; //set the current wafer back to null
            inUse1 = inUse1 + (SimMaster.getSimTime() - startService1);
        }else { // we are working with server 2
            SimMaster.printEvent("S2000 UnLoaded Wafer: # "+ent.getId()+" from Server 2");
            // increment the servers available.
            servers++;
            busy2 = false;
            wafer2 = null; //set the current wafer back to null
            inUse2 = inUse2 + (SimMaster.getSimTime() - startService2);
        }
        // check the queue, non-entity related
        super.scheduleEvent(0,"tryToLoad");
        // unload time is uniform(.66,1.0) hours
        float dt = (float)(.66 + .33 * Math.random());
        super.scheduleEvent(dt,ent,"depart");
    }

    // No need to define the "depart" event here.  Just let the super object
    // take care of sending the entity to the next workstation.

    // unscheduled break down on Server 1
    public void failure1(Object obj) {
        SimMaster.printEvent("Server 1 Failure on S2000");
//        SimMaster.printError("Push \"Resume\" to continue");
        // current entity get destroyed
        if (busy1) {
            busy1 = false;
            SimMaster.printEvent("Wafer: # "+wafer1.getId()+"was destroyed");
            inUse1 = inUse1 + (SimMaster.getSimTime() - startService1);
```

```
        } else {
            SimMaster.printEvent("Server was idle");
            servers--;
        }
        // schedule the returnToService time
        float dt = (float)(1 + Math.random());
        super.scheduleEvent(dt,"returnToService1");
    }
    // unscheduled break down on Server
    public void failure2(Object obj) {
        SimMaster.printEvent("Server 2 Failure on S2000");
//        SimMaster.printError("Push \"Resume\" to continue");
        // current entity get destroyed
        if (busy2) {
            busy2 = false;
            SimMaster.printEvent("Wafer: # "+wafer2.getId()+"was destroyed");
            inUse2 = inUse2 + (SimMaster.getSimTime() - startService2);
        } else {
            SimMaster.printEvent("Server was idle");
            servers--;
        }
        // schedule the returnToService time
        float dt = (float)(1 + Math.random());
        super.scheduleEvent(dt,"returnToService2");
    }
    // fixed and returned to service
    public void returnToService1(Object obj) {
        SimMaster.printEvent("Server 1 was returned to service on S2000");
//        SimMaster.printError("Push \"Resume\" to continue");
        servers++;
        super.scheduleEvent(0,"tryToLoad");
        // schedule next failure
        float dt = (float) (60 * Math.random());
//        System.out.println(dt);
        super.scheduleEvent(dt,"failure1");
    }
    public void returnToService2(Object obj) {
        SimMaster.printEvent("Server 2 was returned to service on S2000");
//        SimMaster.printError("Push \"Resume\" to continue");
        servers++;
        super.scheduleEvent(0,"tryToLoad");
        // schedule next failure
        float dt = (float) (60 * Math.random());
//        System.out.println(dt);
        super.scheduleEvent(dt,"failure2");
    }
    public void reportFinalStatistics() { // called at the end of the run
        super.makeStatisticsWindow(workstationName+" Final Statistics");// (mandatory
line)
        //report the collected stats in the following lines.
        super.printStat("The server 1 Utilization was: "+inUse1/SimMaster.getSimTime());
        super.printStat("The server 2 Utilization was: "+inUse2/SimMaster.getSimTime());
        super.printStat("The current Number in Queue was:"+queue.size());
    }
}
```

APPENDIX F: S1000 vs. S2000 Hypothesis Test

Cycle Time Hypothesis Test Sample Size =5

S1000 Average Cycle Time (x)	59.52900
S1000 Average Cycle Time Sample Variance	0.00445
S2000 Average Cycle Time (y)	50.38570
S2000 Average Cycle Time Sample Variance	0.00434
Pooled Estimator of the Common Variance	0.004395

t critical value for alpha = .01 t(.01,8) = 2.896

H0: The average cycle times are equal: x - y = 0

H1: The S1000 cycle time is greater than the S2000 cycle time: x - y > 0

Test statistic for the test t = 218.11

Result:
Since t > t(.1,8), the null hypothesis can be rejected with alpha = .01.

The S2000 average cycle time is significantly less than the average cycle time of the S1000.

Throughput Hypothesis Test Sample Size =5

S1000 Average Throughput (x)	0.3350
S1000 Average Throughput Sample Variance	0.0004
S2000 Average Throughput (y)	0.3960
S2000 Average Throughput Sample Variance	0.0006
Pooled Estimator of the Common Variance	0.0005

t critical value for alpha = .01 t(.01,8) = 2.896

H0: The average throughputs are equal: x - y = 0

H1: The S2000 cycle time is greater than the S1000 cycle time: y - x > 0

Test statistic for the test t = 4.3140

Result:
Since t > t(.1,8), the null hypothesis can be rejected with alpha = .01.

The S2000 average throughput time is significantly greater than
the average throughput time of the S1000.

APPENDIX G: ProModel vs. NCSOS Hypothesis Test

ProModel vs. NCSOS Sample Size =5 Single Server

Cycle Time

NCSOS Average Cycle Time (x)	59.52900
NCSOS Average Cycle Time Sample Variance	0.00445
ProModel Average Cycle Time (y)	59.41740
ProModel Average Cycle Time Sample Variance	0.12860
Pooled Estimator of the Common Variance	0.06653

t critical value for alpha = .01 t(.01,8) = 2.896

H0: The average cycle times are equal: x - y = 0

H1: The NCSOS cycle time is greater than the ProModel cycle time: x - y > 0

Test statistic for the test t = 0.6842

Result:
Since t < t(.1,8), the null hypothesis can NOT be rejected with alpha = .01.

The NCSOS average cycle time is NOT significantly greater than the ProModel average cycle time.

ProModel vs. NCSOS Sample Size =5 Single Server

Throughput

NCSOS Average Throughput (x)	0.3350
NCSOS Average Throughput Sample Variance	0.0004
ProModel Average Throughput (y)	0.3324
ProModel Average Throughput Sample Variance	0.0006
Pooled Estimator of the Common Variance	0.0005

t critical value for alpha = .01 t(.01,8) = 2.896

H0: The average throughput times are equal: x - y = 0

H1: The NCSOS cycle time is greater than the ProModel cycle time: x - y > 0

Test statistic for the test t = 0.1838

Result:
Since t < t(.1,8), the null hypothesis can NOT be rejected with alpha = .01.

The NCSOS average throughput is NOT significantly greater than the ProModel average throughput.