VIRTUAL FACTORY SIMULATION USING AN ACTIVE OBJECT ORIENTED DATABASE SYSTEM

By

LAKSHMI PRASANNA KUPPALA

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

1998

Dedicated to my Parents,
K.V. Rao and BhagyaLakshmi,
and my Sisters, Padma and Danu
and my Niece, Dolly

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Ali A. Seireg for his constant encouragement and support during the research. I would like to express my gratitude to Dr. Sharma Chakravarthy for his innovative ideas, suggestions, valuable computer assistance and more importantly, for giving me an opportunity to work on this research project.

I would like to thank Dr. John K. Schueller for giving me valuable suggestions and feedback. I would also like to thank Dr. Tufekci for his suggestions and help provided in making this work possible.

This work was supported in part by the Office of Naval Research and the SPAWAR System Center–San Diego, by the Rome Laboratory, DARPA, and the NSF grant IRI-9528390

I would like to take this opportunity to acknowledge my gratitude to my parents and my sisters for their encouragement and inspiration throughout my academic career. I express my gratitude to my friends Sushma, Navleen, Nandini, Raj and Poorab for being there to help me always, and my friends in India, Shanthi, Shobhana, Gayatri, Lalitha and Bhuvana, who have been constantly encouraging me through the research work.

TABLE OF CONTENTS

page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

VIRTUAL FACTORY SIMULATION USING AN ACTIVE OBJECT-ORIENTED DATA BASE SYSTEM

By

Lakshmi Prasanna Kuppala

August 1998

Chairman: Dr. Ali A. Seireg
Co-Chairman: Dr. Sharma Chakravarthy
Major Department: Mechanical Engineering

A manufacturing facility consists of a large number of subsystems like the machining centers, inventory and repair facilities. This makes the overall application a very complicated process with the amount of data-flow being very high. Currently many of the factory-based applications are based on passive databases, where they have to poll the database and take actions once the conditions are satisfied. This means that dynamic event based processes which is very common in a shop-floor application are not supported by the traditional passive databases. Such applications which need timely response have prompted the development of active object oriented database management systems (OODBMS).

This thesis addresses the design and implementation of an event based shop-floor application, using the active functionality of Sentinel. Sentinel is an Active Object-Oriented Database Management System, developed for the purpose of addressing condition monitoring of dynamic events. This functionality is used in applications such as CIM, stock market monitoring and applications that are event-driven.

The implementation presented in this thesis has a user-friendly graphics interface to simulate the shop-floor environment. Intended to be an educational tool, this can also be used for decision making in actual manufacturing facilities by analyzing the results obtained. This tool also facilitates understanding of the concepts like MRP and Kanban. This tool also serves the purpose of analyzing the suitability of

Sentinel for a real-time manufacturing application. To facilitate expansion and porting, a 3-tiered architecture is used to separate the interface logic, application logic and database logic. The front-end interface application has been developed in Java, which is known for its easy portability and expansion features.

CHAPTER 1
INTRODUCTION

1.1 Production Systems

In the broadest sense, a production system is anything that produces something; formally defined as something that takes inputs and transforms them into outputs with inherent value. In manufacturing, the inputs and outputs are usually tangible and the transitions are often physical. The study of production systems involves various components like products, customers, raw materials, transformation processes, direct and indirect workers, and formal and informal systems that organize and control the entire process.

1.1.1 The Flow Process

The backbone of any production system is the manufacturing process, a flow process with two major components: material and information. The physical flow of material can be seen, but information flow is intangible and more difficult to follow. Both types of flow have always existed; however, in the past, information flow was of little importance. As mentioned before, the emerging information technology has provided a way of collecting the information flow and analyzing it to control the material flow to achieve higher throughputs.

Figure 1-1 shows a generic model of physical flow in production systems. Material flows from the supplier to the production system, to become raw material inventory. It then moves to the production floor, where the material conversion process takes place. The material moves through different conversion processes performed at workstations but does not necessarily traverse the same route each time. Material on the production floor is called work-in-process inventory (WIP) [1]. From the production floor the material flows to a location where it becomes finished goods inventory. From there it flows to the customer, sometimes through intermediaries such as distribution centers or warehouses.

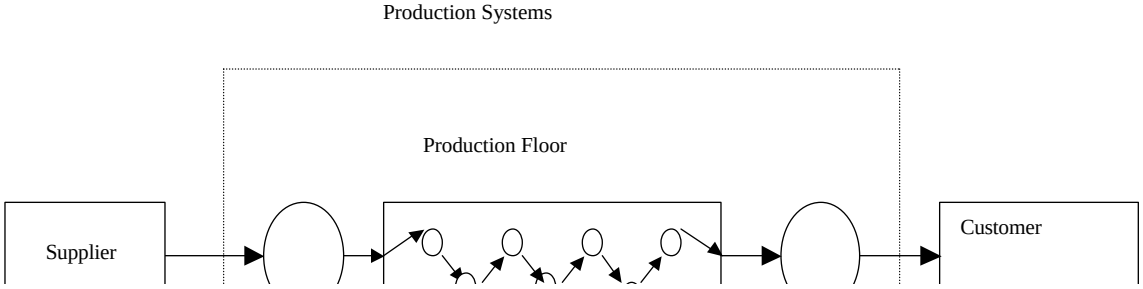


Figure 1.1 Generic Model of Physical Flow in a Production System

1.1.2 Role of Database Management System

A common database services all functions and activities of the production system. Figure 1-2 shows a generic production information system.

There are several advantages of using a DBMS in a production system. Some of them are:
The information regarding the process is easily accessible by all the components such as inventory, shop-floor control and master production control. Changes in one component are easily reflected in the DBMS and can be observed by other components.

- Data compatibility in DBMS simplifies data access and the cost of writing application programs to process information. This way more information can be obtained quickly and easily.
- The static data of the manufacturing unit can be used in the decision-making stage such as selection of material, machine tool and cutting tool.
- The log files can be evaluated dynamically to predict the behavior of a manufacturing unit in terms of efficiency of performance, throughput times, waiting time, inventory etc.

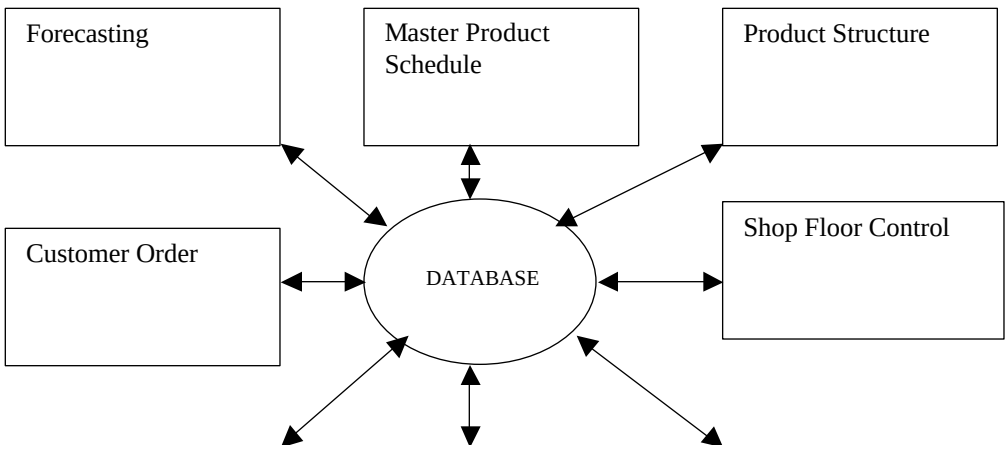


Figure 1.2 Generic Production Database System.

1.1.3 Factory Layout

The goal of production systems is to manufacture and deliver products. The principal activity in meeting this goal is the manufacturing process, in which the material conversion of raw material into a product takes place. The manufacturing process can be viewed as a value-adding process.

The material conversion process takes place on the production floor, which is arranged in a certain way to facilitate flexible as well as optimal conversion. Production volume and product variety determines the type of arrangement, or layout. Fundamentally there are two different types of physical layouts: job shop and flow shop.

A job shop produces low-volume highly customized products. Job shops usually have several elements in common. General-purpose equipment, which can handle, within limits, different types of jobs is normally used. A typical job shop layout of manufacturing equipment is a process layout in which similar machines are grouped. For example, in a machine shop lathes are located in one area and milling machines are located in another area.

A flow shop produces a high volume standardized product. The automobile industry is a good example of a flow shop. Special purpose equipment and little manual skill are required in this kind of layouts. Each product in a flow shop follows the same sequence of operations. A flow shop uses a product layout. Equipment is arranged so that the product always follows the same routing throughout the layout.

Between the extremes of job shops and flow shops there is a batch shop that has flexibility and medium volume production as its characteristic features.

1.2 Technology

The computerized advancement of technology in the manufacturing areas begins with computer aided design (CAD), and includes computer aided engineering (CAE) and computer integrated manufacturing (CIM).

1.2.1 Computer Aided Design

Computer aided design (CAD) is a software system that uses computer graphics to assist in the creation, modification and analysis of design. CAD is used for geometric modeling, automated drafting and documentation, engineering analysis, and design analysis. Geometric modeling uses basic lines curves and shapes to generate the geometry and topology of a part. The part may appear as a wire mesh image or as a shaded solid model. Once an object has been input into the system, it can be manipulated and displayed in a number of ways. The CAD database created from the geometric design includes not only the dimensions of the object but also tolerance information and material specification.

Libraries of design can be accessed so that designer can modify existing designs instead of building new ones from the scratch. Automated drafting produces engineering drawings directly from the CAD database. It also merges text and graphics to produce assembly drawings. CAD is more than a drafter's version of a word processor. Its real power is derived from the ability to electronically link design with other automated systems.

1.2.2 Computer Aided Engineering

Engineering analysis when performed at a computer terminal with a CAD system is called computer-aided engineering (CAE). CAE retrieves the description and geometry of a part from CAD database and subjects it to testing and analysis on the computer screen without physically building a prototype.

1.2.3 Computer Aided Manufacturing

Computer-aided manufacturing refers to the control of manufacturing process by computers. CAD/CAM involves the automatic conversions of CAD design data into processing instructions for computer-controlled equipment and the subsequent manufacture of the part that it was designed for.

1.2.4 Flexible Manufacturing System (FMS)

A flexible manufacturing system is another important technology for production floor operation and control. It also covers the middle ground of the volume variety plot (Figure 1.3), where flexibility is a major requirement. Flexible manufacturing system can be defined as the integration of manufacturing or assembly processes, material flow and computer communication and control. The objective is to let the production floor respond rapidly and economically to changes in its operating environment.

Typical changes in the operating environment concern product mix, production volume, equipment breakdowns, etc. The flexible manufacturing system comprises of technologies of programmable automation, automated material handling, computer control, and communication systems. It is driven by the need for flexibility created by the market-driven environment.

1.2.5 Computer Integrated Manufacturing

Computer integrated manufacturing is a third alternative to medium-volume, medium-variety manufacturing. Computer integrated manufacturing has a broader scope than cellular manufacturing systems of flexible manufacturing systems. Not only is it computer-based, but it implies a high degree of integration among all parts of the production system. All functions of the production are tied into one large computer database, and access to the data is provided to various departments (users) in the organization. Theoretically, materials enter at one end of the plant, and finished products come out the other end, all at the push of a button.

CIM is viewed as a management philosophy that has required technology to implement it. So it can be defined as a management philosophy that uses computers, communication, and information technology to coordinate the business functions with product development, design, and manufacturing. The objective is to obtain a better competitive position by achieving a high level of quality, on-time delivery and low cost.

There is a distinction between FMS and CIM though their definitions appear quite similar. FMS deals basically with the production floor, i.e. local integration, whereas CIM goes beyond the production floor towards global integration. FMS represents more of a bottom up approach to automation, while CIM is top down.

1.3 Push and Pull Models

1.3.1 Push/MRP Systems

Material Resource Planning (MRP) and Push systems are often used interchangeably. Conceptually, MRP can be viewed as a method for the effective planning of all resources of a manufacturing organization. A formal definition is, MRP is a computer based planning, scheduling, and control system [9]. It gives management a tool to plan and control its manufacturing activities and supporting operations obtaining a higher level of customer service while reducing costs.

A due date for each job is determined from the customer's demand or the planned schedule. Jobs are released on a start date, which is the due data minus a lead-time. Hence the lead-time is a deterministic planning parameter. Flow time is the actual time it takes the material to traverse the production system. It is variable, and hence the study of flow time to reduce its variations is very much desired. Once released, the job flows from operation to operation through the production system regardless of what happens downstream. Hence the term Push for this method; jobs are pushed through the production system. Another name for the Push systems is schedule-based systems.

1.3.2 Pull Systems

The goal of a Pull system is to provide a simple production control technique that reduces lead-time and work in process (WIP). In a Push system, a schedule is prepared in advance for a series of workstations, and each workstation pushes its completed work to the next station. With the Pull system, workers go back to previous stations and take only those parts or materials they need and can process immediately. When their output has been taken, workers at the previous station know it is time to start producing more, and they replenish the exact quantity that the subsequent station just took away. If their output is not taken, workers at the previous station simply stop production; no excess inventory/intermediate products are produced. This system forces operations to work in coordination with one another. It prevents overproduction and underproduction; only necessary quantities are produced.

In short, the difference between Push and Pull systems is that a Push system controls work release orders, whereas a Pull system controls the shop floor. To be more precise, Push systems control throughput

(by controlling work release) and measure WIP, whereas Pull systems control WIP and measure throughput.

Kanban, the Japanese word for card, is the tool originally used to achieve these objectives of a Pull system. The Pull technique is also called just-in-time (JIT). There is very little difference between the two terms, JIT and Kanban in that, Kanban is a manual method to implement a Pull system, whereas JIT refers to the whole system, material flow control and management philosophy.

1.3.2.1 JIT

JIT systems combine both the production-control component and a management philosophy. Four basic tenets are required for the success of a JIT system:

- Elimination of waste
- Employee involvement in decision making
- Supplier participation
- Total quality control

Waste is closely related to cost-adding processes. Of all types of waste, inventory has attracted the most attention. Excess inventory covers other types of wastes. The JIT objective of reducing inventory uncovers these problems.

Employee involvement as part of the JIT philosophy is in line with the culture of market-driven systems. In a JIT system this is achieved through teamwork and employee empowerment. Each employee is given more responsibility for the production process.

Supplier participation indicates a different working relationship with the suppliers. Instead of being looked upon as adversaries, suppliers are regarded as partners. The tendency is to reduce the number of suppliers and establish long-term associations with them.

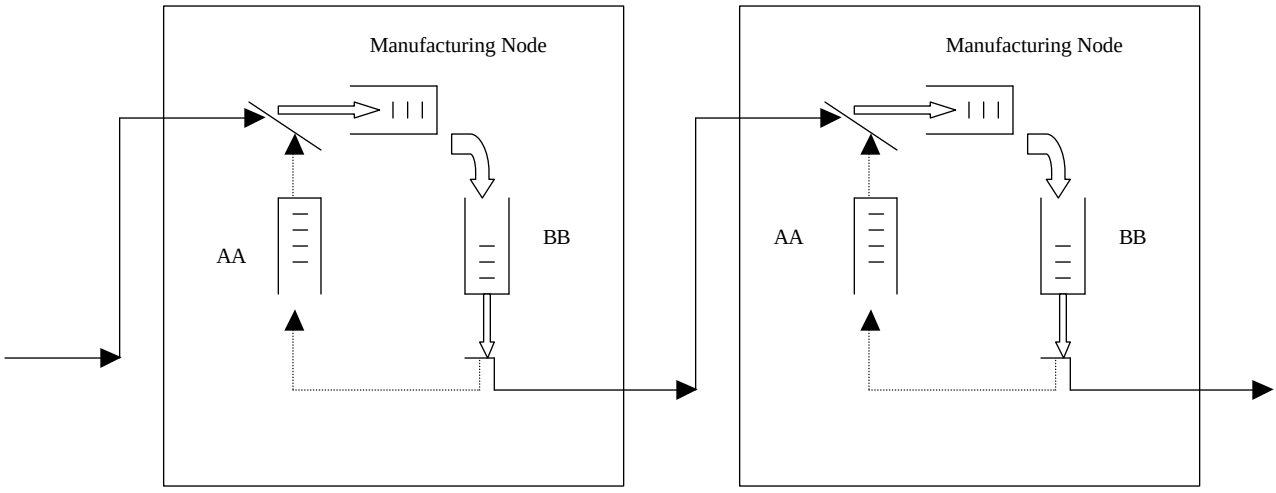
1.3.2.2 Kanban Systems

In Japanese, Kanban means a card or a visible record. In a broader sense, it is a communication signal from a consumer to a producer. As such, it is a manual information system to control production, material transportation, and inventory. There are different types of Kanbans that are implemented in a

production shop [3]. The basic Kanban model used in most of these applications consists of a Kanban cell that has:

- a manufacturing node, containing a server and a buffer where the admitted raw parts await or receive service (the latter is given in order of arrival);
- an output hopper, where completed parts await removal;
- a bulletin board, where requests for new parts are posted.

Every cell has a fixed number of cards or Kanbans, a part must acquire one of these cards in order to enter the cell and must continue to hold it through out its sojourn there. Thus the inventory in a particular cell can never exceed the total number of Kanban cards present in that cell. Any unattached cards in that cell are to be found on the bulletin board and are considered as requests for raw parts. The following figure gives a better understanding of the Kanban activities explained so far.

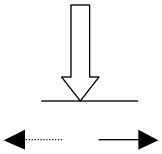


AA Bulletin Board

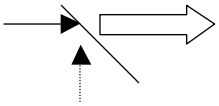
BB Output Hopper

Card Flow

Part Flow



Fork of Part and Card



Join of Part & Card

Figure 1.3 Kanban Cell Model

CHAPTER 2
EMULATION OF MANUFACTURING APPLICATIONS

2.1 Purpose of Emulation

A manufacturing facility is a highly complex system. It consists of a large number of subsystems like the machining centers, inventory and repair facilities. This makes the overall application a very complex process with the amount of data flow being very high. The traditional approach of understanding and improving manufacturing plants has been breaking up of the whole complex system into understandable and easily manageable subsystems. Developing solutions were easier, but when integrated to understand the behavior of the whole system the analysis was not very accurate and easy to perform.

An easy and worthwhile solution was to emulate the whole machine plant and analyze the performance while switching between the various optimization criteria. This not only offers a faster solution but also the ability to interact in real-time.

2.2 Use of Active Databases

The amount of data-flow being very high in a manufacturing facility, a database management system that could efficiently manage data is highly useful. Different kinds of databases and Information Technology have been used so far for manufacturing applications:

2.2.1 Relational Database Management System (RDBMS)

A relational database typically consists of creating two-dimensional tables of data for various entities and relations among different shop-floor control applications. Emulated Flexible Manufacturing Software Developed at the Industrial Research Laboratory, University of Florida, Gainesville, a PC based shop-floor emulation uses RDBMS for the database management of its applications.

2.2.2 Knowledge based System with RDBMS

An expert system recommends a decision based on expert knowledge. An expert system includes a knowledge base, which contains an expert's knowledge on a particular type of problem and a mechanism for reasoning that allows inferences to be made from the knowledge base. Expert systems can be viewed as computerized "consultants" for decision making that use a collection of facts, knowledge, and rules to diagnose problems and suggest solutions. It involves RDBMS for basic storage purposes.

2.2.3 Artificial Intelligence

It attempts to replicate human thought process with computers to diagnose and solve problems. Expert systems can be viewed as one type of artificial intelligence. Other types of AI are neural networks, genetic algorithms, and fuzzy logic.

2.2.4 Object-Oriented Database Management Systems (OODBMS)

This involves the object-oriented concept of representing entities in a real-time application as objects. It provides an expressive modeling framework for complex objects representing various resources used in manufacturing systems. The prime features of object-oriented design are polymorphism, inheritance, modularity and encapsulation. Emulation of manufacturing applications has been done using OODBMS and knowledge based rule specification [4].

All the above alternatives use passive databases, which is a disadvantage for a real-time application such as a manufacturing plant. Each time there is a change in the value of the variables; applications need to update the databases either by polling the database from time to time or by writing an update explicitly. But with complex data involved, a passive database does not offer much of flexibility of maintaining consistency among the databases.

2.2.5 Active Database Approach

Timeliness of event detection and action execution, which is important for a number of real-time applications, has propelled the research needs in active databases. A manufacturing application needs to continually monitor changes to the database state and initiate appropriate actions preferably without user or application intervention. This can be partially achieved by the event-condition-action semantics that are seamlessly incorporated into an OODBMS. The rules in active databases are event-driven and are typically

invoked by operations or state changes caused by a transaction. Active capability is intended as a uniform mechanism for supporting several database functionalities. With all these event-driven features, active OODB seems to offer a better solution to a CIM application. Sentinel an active OODBMS is being currently used for this research work, developed at the Database Research Development Center in the Computer and Information Science Engineering Department at the University of Florida. Sentinel uses the Open OODB as the underlying platform and adopts open system architecture. Applications are written as extensions to the Sentinel (Appendix A).

2.3 Previous Work

Design and implementation of an event-based shop floor control application on Sentinel done by Honnavalli [8] at the University of Florida, Gainesville, provides the basis for this research. The work consisted of event-condition-action rules specified on machine databases and a Motif user interface. The total throughput time for all the batches was calculated and shown as the end-result along with some more statistics of machine breakdown and repair status. Some of the characteristics of that approach are:

- A two-tier architecture with the database in one level and the application along with the user interface in the other level.
- Factory runs as a part of the main thread and a scheduler thread then spawns separate threads for each machine based on the priority attached to each thread.
- The numbers of machines, number of batches, and batch processing times are all taken as inputs. The production status, scrap-count, idle time, and the throughput time are calculated at the end of each simulation.
- Motif was used for developing the user-interface.

Some of the functionalities added to Sentinel to support distribution were not used in the previous implementation and hence had the following limitations:

- A two-tier architecture was used which is not portable and would not scale as well.
    - Scheduling of machines and events has to be taken care of explicitly by means of threads.
- The code was all integrated into one single application to be run on a single host machine, with no distributed feature.

- Extensive testing for large batch sizes was not feasible, as a part of limitation of the user-interface.
- All of the events were primitive events with the events being generated sequentially. The composite and temporal events could not be tested with this application.

2.4 Objective of Present Work

The focus of this work is to test the active capabilities of Sentinel and its suitability to real time applications. It also involves Global Event Detector and the Local Event Detector for event detection, Snoop event-condition-action semantics and User- friendly Java interface. Some of the features of the approach taken in this thesis are:

- A three tier architecture to separate the interface logic, database logic and application logic, that would facilitate porting and expansion easily.
- Flexibility of creating, deleting or just viewing the projects/factory configurations. The specific number of machines, batches and parameters of various distributions to generate the processing times for each batch on an individual machine can either be selected by the user or the default values can be used.
- User-friendly graphics interface that facilitates selection of an existing project and then interacts with the server side applications providing flexibility of getting the status reports at any point of time.
- The application uses the advantages of a distributed environment and can start the manufacturing machines, each on a different host machine.
- Global Event Detector (GED) [5], an important feature of the Sentinel, which can modularize a complex application like that of a manufacturing plant is used with this application. The Global Event Detector helps in detecting global events and communicates them across different machines.

2.5 Intended Use of the Tool

The main purpose of the tool is to test the suitability of Sentinel, an Active OODB, for the real time applications and problems. So a Virtual Factory implementation which is very close to the real world shop floor provides a good example to test out all the suitable capabilities of Sentinel. This tool is also

intended to serve the purpose of users in understanding some of the basic and important manufacturing concepts.

Users can first build their own configuration by specifying the number of machines, number of batches, and selecting the machines from the machine database based on the machine characteristic features. They can then see the manufacturing simulation at runtime and understand the basic flow of jobs from machine to machine and then analyze various results like the idle time of a machine, total throughput time of a particular batch etc. They can also change the batches to a Kanban production line and then compare the results with the previous ones. The users can visualize the manufacturing process in a better way as the Machine Applications in this tool are executed on different host machines, thus giving a feel of distributed machines.

So the basic purpose of this tool is to provide a means of specifying or creating some factory configurations and then at the runtime view the basic flow of the jobs through the machine and study the throughput time for different configurations. In doing this, Sentinel is to be used to study its adaptability for real-time applications.

CHAPTER 3
DESIGN OF THE VIRTUAL FACTORY

3.1 Architecture

The key issues in designing any application are Modularity and Portability. These can be achieved by using layered structures and portable languages.

3.1.1 3-Tiered Client Server Architecture

When compared to the traditional 2-tier architecture of database logic and application/interface logic, the 3-tier architecture facilitates the separation of interface logic from the application logic. This helps in easy porting and expansion of the application with a better understanding of the application nuances.

Figure 3.1 illustrates this architecture. In this architecture, the interface interacts with the application logic via standard protocol like socket (TCP/IP) or RPC. The application logic interacts with the database server via standard database protocols. The application logic receives requests from multiple clients on the interface side, translates them into database queries, and passes the queries to the database server. It then sends the requested information back to the interface. Since the separation of interface and application logic results in modularity it is easy to expand individual modules. The architecture results in the reduction of development and maintenance cost [6].

This architecture has been adopted in the present implementation because of the advantages discussed above.

3.1.2 Use of Java as a Portable Language

Java has exceptional features that can be listed as follows [2]:

- Simple: Java has the bare bones functionality needed to implement its rich feature set. It does not add lots of syntactic sugar or unnecessary features.

- Object-Oriented: Almost everything in Java is a class, a method or an object. Only the most basic primitive operations and data-types (int, for, while, etc.) are at a sub-object level.
- Platform Independent: Java programs are compiled into a byte code format that can be read and run by interpreters on many platforms including Windows 95, Windows NT, and Solaris 2.3 and later.
- Safe: Java code can be executed in an environment that prohibits it from introducing viruses, deleting or modifying files, or otherwise performing data destroying and computer crashing operations.
- High Performance: Java can be compiled on the fly with a Just-In-Time compiler (JIT) to code that rivals C++ in speed.
- Multi-Threaded: Java is inherently multi-threaded. A single Java program can have threads being executed concurrently.

With all such rich characteristic features, Java is one of the most suitable languages for the interface used in this research work. Also Java classes can connect indirectly to a database server by opening a socket connection to an intermediate server, which in turn opens a connection to the database. Though this method of communication is not the most direct solution, it is portable since the middle-ware Proxy [6] can translate between platform dependent database calls and a custom platform-independent socket based protocol. This is the way the application elements have been organized in the 3-tier structure. Proxy server can be illustrated clearly in figure 3.2

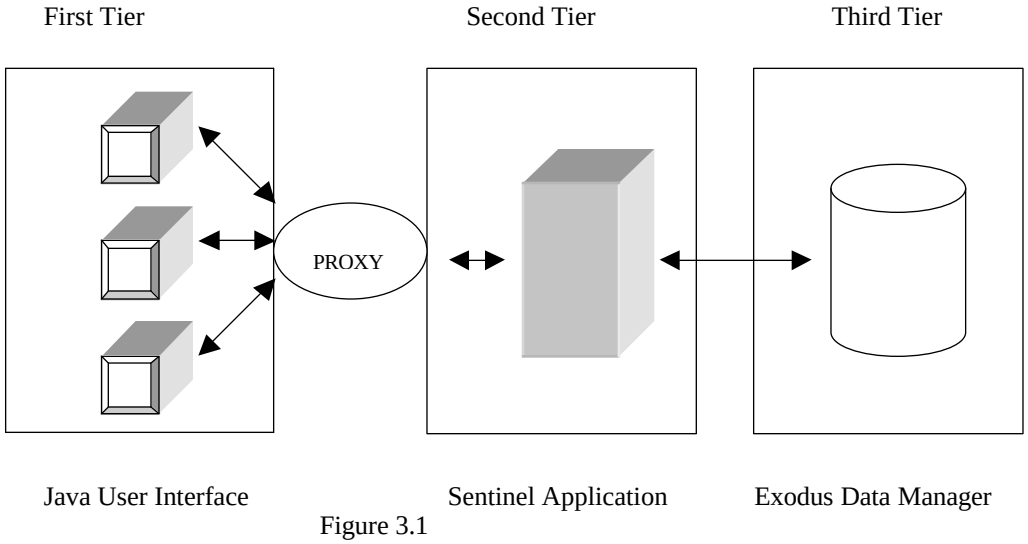


Figure 3.1

3.1.3 Proxy Server

The literal meaning of the word Proxy is *an agent authorized to act for another.*

Here the proxy is the socket protocol used in between the first tier and the second tier. In middle-ware terminology, a proxy that communicates between a client and server is called a gateway. One Proxy can easily service multiple servers by using multi-threading. So in this Virtual Factory Application, a proxy server is used to communicate between multiple Java clients and the server side application. This facilitates a web-based version of the application with literally no changes in the architecture.

3.2 Storage Manager

Sentinel applications use Exodus Storage Manager [12] to manage their stored objects. Exodus Storage Manager is a multiple-user object storage system supporting versions, indexes, single-site transaction, distributed transactions, concurrency control and recovery. It has client-server architecture. An application program that uses the storage manager may reside on a machine different from the machine or machines on which the storage server or servers run. The storage manager server is a multi-threaded process providing asynchronous I/O, file, transaction, concurrency control, and recovery services to multiple clients. It stores all data on volumes which are either Unix files or raw disk partitions. It provides objects for storing data, versions of objects for grouping related objects, and indexes for supporting efficient object access. The Virtual Factory Application uses the Exodus storage manager to manage its stored objects. The various databases used in this project are explained in later chapters.

3.3 Design and Architecture of the Virtual Factory

The Design of this project consists of two basic modules, a Project Configuration and the runtime Virtual Factory Simulation.

3.3.1 Project Configuration

A Project configuration consists of creating or deleting project/factory layout. The only database that is required by this module is the machine database.

Machine database is an extensive database in that it has all the details of all machines to be considered for emulation. The Machine database consists of all the machine details like the machine name,

Machine Id, Machine Characteristic features, Machine Status etc for all the machines that are present in the factory. The Machine database is extensively managed by the supervisor. The application developer here plays the role of the factory supervisor and can make changes to the database. The user does not have access to change the Machine database. So the Machine database is created separately and populated.

The user can create a required configuration or layout by making use of the machines that are in Machine database. Project Configuration layout is a separate module that interacts just with the database and does not interact with the runtime simulation. The configurations selected by the user are persisted as named sets and stored in the database They are retrieved from the database at the runtime of the factory simulation. So at the runtime the user need not worry about creating new configurations, rather the user has to just select the configurations that have been already created and stored in the factory database.

3.3.2 Runtime Factory Simulation

The Virtual Factory Application has been sliced into smaller modules for easy implementation and understanding.

3.3.2.1 Control/Factory Application

A Control/Factory Application is more like a report generator interacting with the front-end and passing on information about the status of the machines, batches, repairs, etc. Control/Factory Application is a Sentinel application that has the capability of communicating with the other Machine Applications by means of events and subscriptions.

3.3.2.2 Machine Application

A Machine Application constitutes the rules pertaining to certain machines and does not interfere with the factory rules. The important design features are:

- The Machine Application is designed to act as a generic machine with all the common functionality of a generic machine embedded into the application.
- Every machine has the same generic functions such as processing a specified batch, start production, stop production and sending for repair if machine breakdown occurred.
- The other details that are specific to an individual machine like the machine status, machine ID, machine size, machine features are stored in the machine database, which is used for creating Project

configurations. At the runtime the selected project configuration that has all the details of machines that were selected for the specific configuration is fetched from the database. This specific machine information is used at the runtime.

So the Machine Application is so designed to separate the generic functionality of a machine from the specific characteristic details by storing the specific details separately in the database and using it at runtime. This single application with all the general machine events is used to act as different machines during the simulation.

3.3.2.3 Java User Interface

As discussed previously, the 3-tiered client-server architecture is used and hence the user interface is separated from the Sentinel application logic. The user-interface has been developed using Java frames. There are two main applications:

- The Report Generator: The Report Generator is designed to display the various status reports of the Factory. This application acts as a client to the proxy server and displays the status reports only when requested by the user. It provides an initial communication with the Launcher Application and the Control Application, thus providing the user with the choice of selecting among the various project configurations that already exist in the database.
- The Machine Display: This application is designed to simulate the various batch operations in the shop floor. It shows all the sequential steps of processing the batches. This Java application is a server application that is constantly listening for client requests, once a client connection is established it spawns a thread to handle the client request and the main process is ready to accept the other requests from the other clients. The multi client server architecture is followed here.

3.2.2.4 Launcher Application

This application does the initial communication with the front-end Java Client and executes the script to start the required number of Machine Applications, control applications and the other required event-detector servers. To facilitate easy porting of the application to the world-wide web, this Control Application has been introduced.

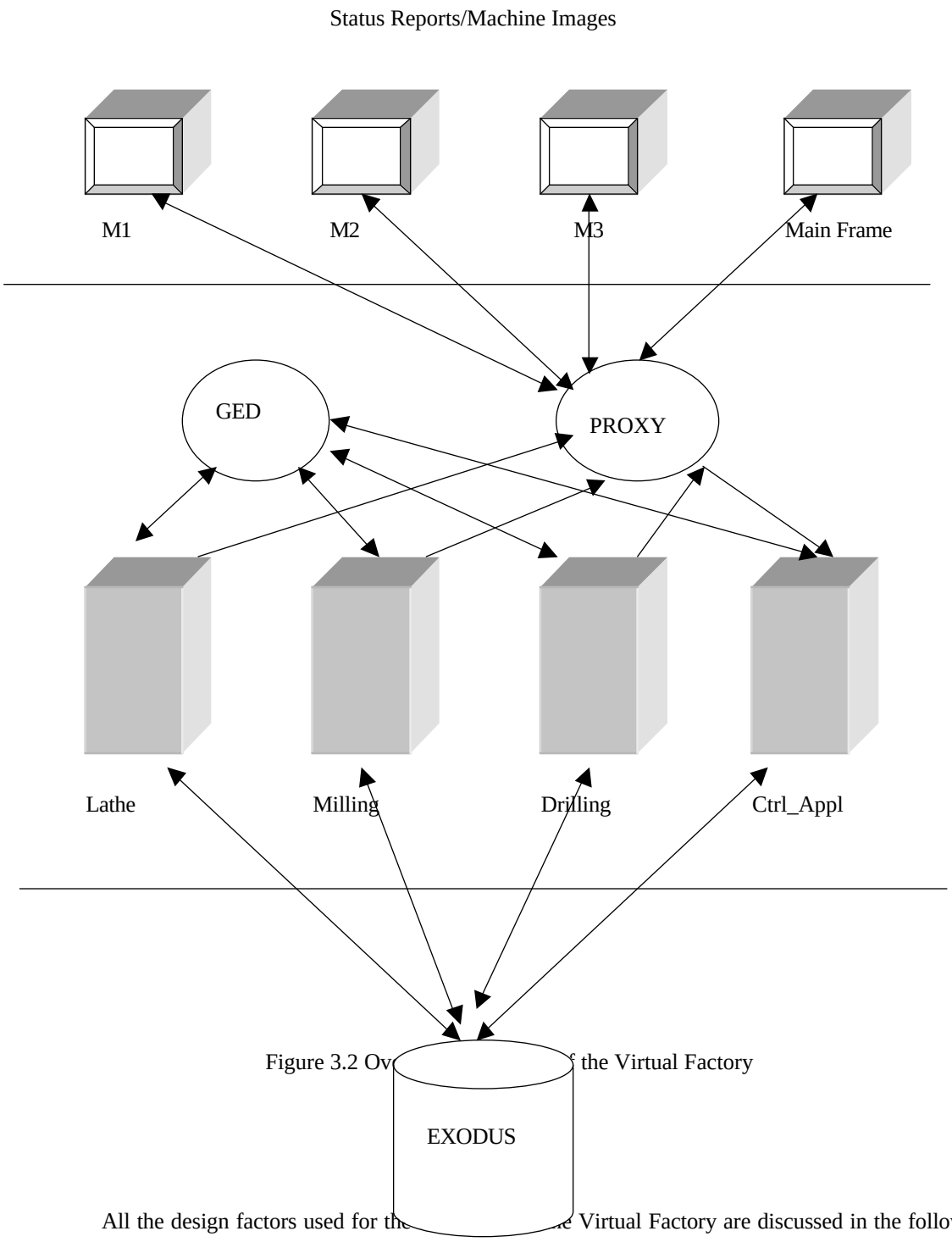


Figure 3.2 Ov f the Virtual Factory

All the design factors used for th e Virtual Factory are discussed in the following sections. The Figure 3.2 gives a better understanding of the overall architecture.

3.3.4 Data Flow in a Manufacturing Process

Typically in a manufacturing unit a central database handles the data management for all the components such as supervisory databases, machine tool databases, repair databases, inventory, etc. The factory layout can vary between job shop and flow shop. Here for simplicity the job shop layout that is more like in-line production is considered. So the job flow is from one machine to the next machine in line. This is the basic design that is used for a job flow in the simulation process. The basic data flow can be seen in Figure 3.3

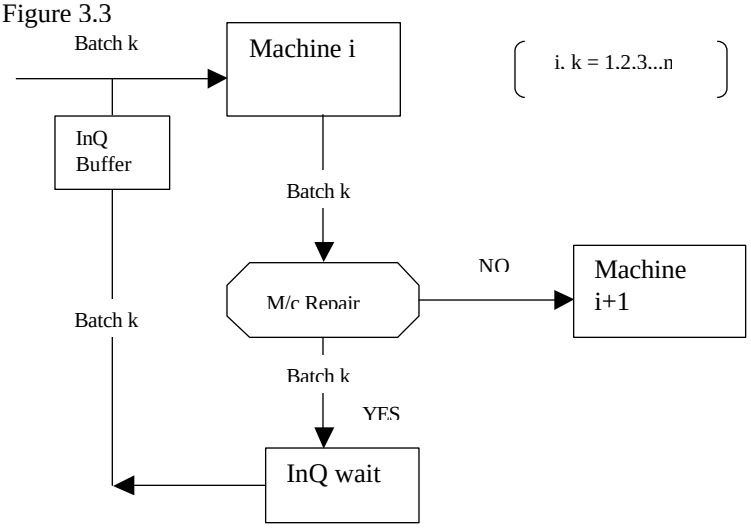


Figure 3.3 Modules of Virtual Factory

3.3.5 Data Structures

The main schema consists of the machine class, factory class, batch class, and a repair class. The inventory details are not put into a different class to keep it simple. They are handled in the machine and the factory classes.

The factory class consists of methods such as start-production, stop-production, get-status-reports which when called appropriately triggers the corresponding rules and performs the actions accordingly. Similarly, the machine class consists of methods such as machine-production-begin, machine-production-stop, batch-begin, batch-stop, and machine-breakdown. The methods start-repair, stop-repair are included in the repair class. These methods along with the regular methods for getting and setting the parameters of a specific class serve as a complete module. Together these form the complete schema of the application.

3.3.6 Rule Specification

A rule in Sentinel consists of an event, a condition and an action specifications [10]. Events can be of two types: a method level event or an instance level event. The begin and end of an event is indicated by *begin(eventName)* and *end(eventName)*. An example of the semantics is shown as follows:

```
Class Machine: public REACTIVE {
private: char Mac_Name[MAXLENGTH];
         int  Mac_ID;
         char Mac_Status[MAXLENGTH] ;
         Set_Batch *batches ;
         Set_Repair *repairs ;
public: Machine(char* ,int ,char* ,Set_Batch* , Set_Repair* );
         Machine(char* ,int ,char* );
         Machine();
         Machine(const Machine &);
         ~Machine(){};
         char* get_mac_Name();
         int   get_mac_Num();
         void set_mac_Status(char *);
         char* get_mac_Status();
         void print();
         FILE *print(FILE *);
         Set_Batch *get_batches();
         void set_batches(Set_Batch *);
         Set_Repair *get_repairs();
         void set_repairs(Set_Repair *);
         void Delete_BatchRepair();
         event end(MacineBegin) void MachineProductionBegin();
```

rule MachineBegin_Rule[MachineBegin, MachineBeginCondition, MachineBeginAction, RECENT];

}

The event shown above occurs when the MachineProductionBegin() method is called, after completion of the executable statements of the method. Once the occurrence of the event is detected by the event-detector, the rule MachineBegin_Rule is triggered and if the condition function evaluates to true the action function is executed.

There are many such events that are defined over the various manufacturing functions. Events can be primitive or composite. Primitive events occurrence is notified after the beginning or at the end of the method call on which the event is defined. This application has a lot of primitive events defined for the most basic operation of manufacturing firm that are as follows:

- Start production at a Factory
- Stop production at a Factory
- Show Machine status reports
  - Start a Machine
  - Stop a Machine
  - Start a Batch (defined on the machine class, so that each machine has a batch-begin for a batch on that particular machine)
  - Stop a Batch
  - Machine breakdown, sent to Repair.
  - Repair start for a particular Machine
  - Repair stop for a particular Machine

The Snoop language provides various operators to express composite events as a combination of two or more primitive events and temporal events, which are detected with respect to time elapse after the occurrence of a primitive event. For example: a plus operator is defined as

e2 = e1 + [5sec];

The second event e2 is a temporal event and is detected after 5 seconds after the occurrence of the first event e1.

In the present application there are temporal events like batch-stop (batch-stop should be detected after the batch_start + the machining time). Similarly the repair-start and repair-stop events are temporal events.

The events can also be classified as local events or global events. Local events are defined on the method local to the class they are referenced in whereas global events depend on the occurrence of another primitive event in a different application. Since each machine is modeled as a different application to be run on the same or a different host-machine, global events become necessary in this application. An example of a global event specification is shown as follows:

Class Machine: public REACTIVE {

Public:

event BatchBegin = Machine_BatchDone::host__appname;

rule BatchBegin_Rule[BatchBegin, BatchBeginCondition, BatchBeginAction, RECENT];

}

In the above example a BatchBegin event of a particular batch occurs on a given machine (say Machine2), only when the BatchDone event of that particular batch on the previous machine (say Machine1) is notified to Global Event Detector [Appendix-A]. This is specified by way of the semantics that the application name (Machine1) and the host-name (for example: mango.cise.ufl.edu) where the application are specified in the event specification. Once the global event detector gets the notification about the global event, it waits for the primitive event on Machine1 to occur and then notifies Machine2 of the occurrence so that the next event can take place. Most of the local events defined in the Control Application are subscribed to, by the Machine Applications as global events. The global events are typically the report generation events.

3.3.4 Databases

A factory consists of a set of machines and each machine processes a set of batches. So there is a factory database which consists of the factory name, user name, and the data of creation.

A Machine database consists of various attributes such as the machine name, length, width, size, status, etc. A Batch database consists of the various distribution details required to calculate the processing times and the batch identification number. Similarly, a Repair database has the distribution details required to calculate the time of repair and the type of repair.

Machine database is an extensive database in that it has all the details of all machines to be considered for emulation. This has extensive protection in that it cannot be altered by the user except for the application software developer similar to that of a supervisor in a manufacturing plant.

The other databases like those of batches and repair consist of only default values to be used, in case the user does not want to select new values. The factory database is created by the user at the run time.

CHAPTER 4
IMPLEMENTATION OF THE VIRTUAL FACTORY

The implementation details of the two main modules, the project/factory configuration module and the runtime Virtual Factory simulation module are discussed in detail in this chapter.

4.1 Project/Factory Configuration

The Factory configuration module consists of a Configuration Server and a user-friendly Java interface. The implementation details are discussed in the following sections.

4.1.1 Configuration Server

The Configuration Server is a C++ process running on a Solaris operating system. The Configuration Server's main purpose is to provide the users with the flexibility of creating their own factory configurations. The functionality of the Configuration Server can be summarized as:

- Initializing the Factory/Project database.
- Providing clients/users with the existing project's information and the default parameter lists for the batches and repair objects.
- Facilitating the insertion and deletion of project configurations. A user can add a new project configuration or delete an existing one. All the parameters for batches and repairs can be specified individually or given the default set of values.
- Providing clients with the available machine information from the Machine database, for selection. The user cannot alter the machine database, as it has all the machines that are available for factory production. Only the supervisor has permission to alter the Machine database, in this case the supervisory role is assumed by the application developer. The Machine database has all information that could aid the user in the selection process.
- When a commit is received, all the information of machines, batches etc., selected for that particular configuration is written into the database and used when required.

When the Configuration Server is first invoked it would check for existing project/factory database and if it is not present would create a new database with a few sample projects. It then provides the options of creating a new configuration, deleting an existing configuration or just viewing the existing configurations. Based on the selection, it provides the further choices of selecting number of machines, batches, machines, batch details, repair details etc. The different batch parameters can be either chosen different from the default values or could be the same.

The queries are written using Object-Oriented Query Language (OQL) [6]. The query used to retrieve all the machine information from the existing machine database is written as:

```
Query {
    Machine-set = SELECT * FROM Machine machine IN Machine_DB;
  }
```

Machine_DB is the name of the machine database and the type of table/class to be retrieved is Machine class. The Machine_DB cannot be altered by the user. Only the supervisor can add or delete machines to the machine database. This is required to simulate the real time production situation in which only the higher authorities can add machines to the shop floor and not the operator.

The other queries that are of importance are the insert and delete queries. When a user specifies the number of machines, batches and other details required for the new configuration, by means of the front end, and commits it, the new configuration is written to the factory database by name Factory_DB and is stored there. When the User needs to delete a specific configuration, the delete query is used to delete the configuration, identified by means of the configuration name, from the factory database, Factory_DB.

```
Query {
  INSERT
  Factory( FacName, FacDate, FacUser ,macset)
  INTO Factory-DB;
}
Query{
  DELETE FROM Factory factory IN Factory-DB
  WHERE (!strcmp(factory.get_FacName(), facname));
```

```
}
Query {
    factory_set = SELECT * FROM Factory  factory IN Factory-DB;
}
```

A TCP/IP socket connection is used between the Configuration Server and the Java side user-interface client application. When a client requests for a service, the server forks a child process to serve the client. The server provides message-driven services to the client and thus creates a very user-friendly interaction.

4.1.2 Java Interface for the Project/Factory Configuration

The Java Interface has a set of windows that are sequentially arranged to show up so that the user gives in all the information required and sees the desired output properly. It has Java Socket calls written in a class called VirtFacSocket.java, the instance of which is globally used to establish connection with the Configuration Server. The Java application here only acts as a client to the proxy server, sending and receiving messages from the proxy server.

The error handlers have been added to care of the exceptions so that the user does not give any irrelevant information. The different frames can be seen in Appendix B.

4.1.3 Sample Factory/Project Configurations

Some of the sample configurations that were created by executing this module are shown here. The configuration consists of the number of machines, number of batches, batch processing time parameters and the repair processing time parameters. These configurations are stored in the database and used during the run time of the Virtual Factory simulation. The individual frames of the Java front-end are shown in the Appendix B. For better understanding a sample Factory configuration is shown in Figure 4.1

Factory Name: DB_VirtFac
Created on : 05/19/98
By : Sharma

Number Of Machines : 4
Number of Batches : 4

Machine Details

Mac_Name	*Mac_ID*	*Mac_Staus*
Lathe	*1*	*Up*
Milling	*2*	*Up*
Drilling	*3*	*Up*
Finishing	*4*	*Up*

Batch Processing-Time parameters

Bno.	*Distr*	*p1*	*p2*	*IsKanban*	*NumKanban*
1	*Exp*	*0.2*	*0.3*	*No*	*1*
2	*Exp*	*0.1*	*0.4*	*No*	*1*
3	*Exp*	*0.4*	*0.1*	*No*	*1*
4	*Exp*	*0.5*	*0.2*	*No*	*1*

The user can edit any of the batch processing parameters or take the default values.

4.2 Virtual Factory Implementation

This distributed Virtual Factory Application utilizes the ECA rules in a real life situation (distributed environment) with GED and OQL usage in conditions and actions. The GED handles event requirements of different component applications for supporting distributed monitoring. The GED uses the consumer/producer paradigm for receiving and sending events and supports both global composite and locally composite events. The architecture of the Virtual Factory is the exact three-tier model where each level is separated from the other very distinctly, which has been discussed in the previous chapter. A proxy server is used to mediate between the Java user-interface application and the database server-side

applications. The Virtual Factory is composed of the many modules that are discussed in the following sections.

4.2.1 Control Application

Control Application simulates a status report generator that can send status reports to update the databases. This application is responsible for dispatching the incoming reports/requests to the concerned application by raising local events which are in turn subscribed to, by the other applications through Global Event Detector. For example, when the Java Client requests the production start event, the Control Application raises its local event production-start, which in turn notifies the Machine Applications that are subscribed to this event, and in turn raises their respective machine-start events. So the communication between the Sentinel applications and the user interface applications is mainly by the control application. The Control Application receives all the requests of the Java clients and through GED, the requests are sent to the appropriate application for further updates.

The Control Application has the basic schema consisting of factory, machine and batches. It has rules written in the factory class, all of which are based on primitive events. The primitive events are raised when the control-application receives requests from the user interface. Once the events are raised the other application that are subscribed to these primitive events are notified by the GED and the corresponding Machine Application's global events are raised. The events that are defined as a part of the Factory class definition are shown here, the other details of the class members are not shown here:

Class Factory : public REACTIVE {

private:

---

public:

---

event end(FacBegin) void Fac_Prod_Begin();

rule FPB_Rule[FacBegin, Cond_TRUE, FPBAction, RECENT];

event end(FacStop) void Fac_Prod_Stop();

rule FPS_Rule[FacStop, Cond_TRUE, FPSAction, RECENT];

event begin(FacRep) void Fac_Report();

event begin(MacRep) void Mac_Report();

event begin(BatchRep) void Batch_Report();

event begin(RepairRep) void Repair_Report();

rule FacRepRule[FacRep, Cond_TRUE, FacRep_Act, RECENT];

rule MacRepRule[MacRep, Cond_TRUE, MacRep_Act, RECENT];

rule BatchRepRule[BatchRep, Cond_TRUE, BatchRep_Act, RECENT];

rule RepairRepRule[RepairRep, Cond_TRUE, RepairRep_Act, RECENT];

};

The events like machine-production-start and machine-production-stop are raised when the front-end Java client sends a request of production-start or production-stop to the Control Application through the proxy server. The event occurrence is then notified to the global event detector so that the other applications which have events subscribed to these local events raise the corresponding global events like machine-start and machine-stop and thereby communicate and send information back to the front end. The other events are the report generator events. When the user requests for the machine status reports the request is sent to the Control Application and the Control Application raises its machine-report event and thereby raises the machine-report global event of the individual application.

4.2.2 Machine Application

Machine Application is one of the main applications that keep track of the status of machines, the repair conditions and the batch operations. The number of Machine Applications to be started is determined at the runtime, depending on the project/factory selected and accordingly the Machine Applications are started. For example, if a project selected has four machines in its configuration, four Machine Applications are started with different application names. As explained in the design phase the Machine Application is a generic application that has the generic functionality of any machine. So the Machine Application has the capability of acting as any specific machine as the application is separated from the machine dependent characteristic features. These features are separately stored in a machine database.

Since the bare-bone structure of any machine is identical to any other machine, the same schema of machines is used for all different machines. The Schema of the Machine Application with the event definitions is shown here:

```
class  Machine : public REACTIVE
{
private:
----------
public:
----------
event MacB = Factory_FacBegin::rain__CtrlApp;
rule MacB_Rule[MacB, MacBCond, MacBAction, RECENT];
event MacStop = Factory_FacStop::rain__CtrlApp;
rule MacStop_Rule[MacStop, MacStopCond, MacStopAction, RECENT];
event FacReport = Factory_FacRep::rain__CtrlApp;
rule FacReport_Rule[FacReport, ReportCond, FacReportAction, RECENT];
event MacReport = Factory_MacRep::rain__CtrlApp;
 rule MacReport_Rule[MacReport, ReportCond, MacReportAction, RECENT];
event end(BatchDone) void Batch_Done(int CurrBatch);
rule BatchDone_Rule[BatchDone, BatchEndCond, BatchEndAction, RECENT];
event end(BatchBegin) void Batch_Begin();
rule BatchBegin_Rule[BatchBegin,BatchBeginCond BatchBeginAction, RECENT];

event BatchB = Machine_BatchDone::host__appname;
rule BatchB_Rule[BatchB, BatchBCond, BatchBAction, RECENT];

event begin(BreakDown) void Mac_Breakdown();
event StartRep = BreakDown + [5 sec];
rule StartRep_Rule[StartRep, StartRepCond, StartRepAction, RECENT];
event begin(RepDone) void Rep_Done();
rule RepDone_Rule[RepDone, RepDoneCond, RepDoneAction, RECENT];
        };
```

Most of the events in the Machine Application are global events, i.e., they wait for the local event occurrence in the other applications to be notified to GED. The events MacB (Machine-begin), MacStop (Machine-stop), FacReport (Factory-status-reports), MacReport (Machine-status-reports) are the global events that are raised only when the corresponding events FacBegin, FacStop, FacReport, MacReport of the Control-application running on a host machine ("rain.cise.ufl.edu", in the above rules) are raised. When the local events of the Control Application are raised the event occurrence is notified to the GED and the GED sends the notification to the Machine Application after which the global events are raised and accordingly the rules are triggered.

The other important global event is the BatchB (Batch-begin). This global event is subscribed to the BatchDone event of the previous Machine Application running on the host machine by name "hostname". Since a single generic Machine Application is used for all the machines, the hostname and machine-application-name are not hard coded. Instead at the runtime the global file which consists of the global event specification is changed at the runtime with the appropriate hostname and the application name. The example code of the global file is shown above:

```
0 Machine_MacB 1 * global rain__CtrlApp Factory_FacBegin *
0 Machine_MacStop 1 * global rain__CtrlApp Factory_FacStop *
0 Machine_FacReport 1 * global rain__CtrlApp Factory_FacRep *
0 Machine_MacReport 1 * global rain__CtrlApp Factory_MacRep *
0 Machine_BatchB 1 * global host__appname Machine_BatchDone *
```

The last line of the global file shows the generic BatchB event, which waits for the BatchDone event occurrence of the Machine Application with application name "appname" and host name "host". These names are changed at the runtime, any Machine Application can run as lathe, drilling or milling as the user wishes, and the machines can be run on different host machines like rain, orange etc.,

One single executable of Machine Application is called with different names without having to make multiple copies of the same executable. This is provided by setting the environment variable APPLCONFIG to different files with different application name. For example, an environment-configuration file would look like:

```
::APP_NAME lathe
::GED_HOST manatee
::GLOBAL_FILE global_1
::RESUME init
::port 6000
```

The first line specifies the application name such as lathe, milling, drilling.These files are generated at the runtime, after the user has selected a specific project. Based on the project selected, the number of machines in that configuration are determined and accordingly that many environment-configuration files are generated with the application-names set to the user selected machine names. The global files are then manipulated to take in the correct hostname and application name.

For example, after the global files are set properly, batch-begin event of a particular batch on milling machine would wait for the occurrence of batch-done event on the lathe machine running on "rain.cise.ufl.edu". This way, the sequence of all batches is set.

The other events are the repair events. A random number is generated to determine if the machine is under breakdown condition or not. Based on the condition, the repair event (temporal event) is raised after a certain amount of time (defined in the event) has elapsed.

The Machine Application simulates most of the production activities such as machining time and total-processing times of individual batches. The various batch parameters like the distribution name, distribution parameters, Kanban facility, number of Kanban cards etc., are all fetched from the project database. By default the MRP concept is utilized, like a certain inventory is taken as input in the form of batch numbers and then the batches are processed in line. If the user has selected a Kanban line and more number of cards (more than one), the production line assumes a Kanban production with minimum inventory, that is the number of batches selected by the user. Depending on the number of cards, the machining times vary and the more the number of cards on each machine, the better the throughput time results are. Whenever a batch is processed the total through put time of that particular batch is calculated and kept in memory and so are the other calculations.

The in-queue and out-queue status are all noted down and all the actions are written into a LOG file. The LOG file is further analyzed to get the various desired results and analysis.

Since a single Machine Application is called to act as different machines in the project, the application takes care of differentiating between the different machines and act accordingly.

4.2.3 Launcher Application

A Launcher Application is supposed to be running on the same machine as that of the interface Java application, so that the initial communication can be set up. This application initially communicates with the front-end and retrieves the selected project information from the database. It then executes a script to start the GED server, proxy server, Control Application and Machine Applications (equal to the number of machines present in the selected project). The script accordingly generates the environment-configuration files with proper application names and then changes the global file's host names and application names. All this is done at the runtime. A TCP/IP socket is used to establish communication between the launcher and Java user-interface. It is just another server program written to establish initial setup and also for easy porting to a web-based environment.

4.2.4 Java Interface

Java applets and Java graphic applications mainly deal with Abstract Window Toolkit commonly referred to as the AWT [7]. It provides a generalized set of classes mainly consisting of the following:

- Component class: It is an abstract class for GUI components such as menus, buttons, labels and lists.
- Container class: It is an abstract class that extends Component. Classes derived from Container, most notably Panel, Applet, Window, Dialog, and Frame, can contain multiple components.
- Graphics class: It is an abstract class that defines methods for performing graphic operations in a component. Every component has an associated Graphic object.
- Layout-Manager interface: It is an interface that defines methods for positioning and sizing objects within a container.

All the classes specified above have been used for the Java interface application, which has been developed for a virtual factory.

A complex window design makes the interface difficult to manage and expand. So the interface logic in this application has been to have a set of windows and then assemble them as required. This kind of design has an added advantage that every window will show up properly on various platforms and various resolutions. Each window is implemented by a Java class inherited from Frame class in AWT.

The user-interface for the Virtual Factory simulation consists of two parts: Java Client Jfac and Java Server application MacShow.

4.2.4.1 Java User-Interface Client Application

The user interface first provides the factory/project configurations that have been stored in the database using the Configuration Server application. The user is expected to create and store the required configurations even before he starts the Virtual Factory front-end. On selecting a particular project, the user is given the flexibility of choosing different status information like overall factory status-report, all machine status-report, all batch status-report and repair report. The machine status reports also has images of machines that are shown along with the status reports of the machines, to simulate a manufacturing shop-floor. The status reports for all the machines are constantly shown in separate machine windows, the data shown being updated from time to time. There is a flexibility of displaying the status reports of factory and machines, at the user's request.

The reports and the corresponding windows are shown when the user selects a specific request. The different requests that can be served are

- Factory Status reports
- All the Machine Details and Status
- The Repair status

Only one class *Jfac* has a main function defined and the other class objects instantiated and called within this class. So the Java interface can be run as a stand-alone application or it can also be run as an applet if required. The Java application acts as a client alone and not a server. It is a client that sends and receives messages from the proxy.

4.2.4.2 Java User-Interface Server Application

In this part of the User-Interface a server application is developed. The class MacShow (Machine-Show) is mainly developed to show the batch flow among the machines from time to time, without the user requesting for it. So the server process constantly listens for client requests from the proxy (proxy acts as a client in this case). Everytime a batch finishes its operation on a particular machine it send a message to the proxy server and the proxy server establishes connection with the Java server and sends the message. This information is displayed so that the user gets a better idea as to how the batches are flowing. When the

server application establishes a connection with the proxy client it runs a thread to handle the client request and the main thread returns back to handle further requests by other clients. As explained in the architecture, it is a multi-client server architecture.

The Java User-Interface Server Application is a separate application and is started by a script in the Launcher Application later. Once the production-start is requested by the user in the Java-Client Application the corresponding batch flow is shown in the frames of the Java-server application.

4.2.5 Proxy

Proxy is a server that acts as an intermediate source of communication between the Java Client and the Sentinel applications. It opens a TCP/IP socket and waits for the client requests in an infinite loop. Once the connection is established between the Java client and the proxy or the Sentinel application and the proxy, the main thread forks off and the child thread handles the request while the control of the main thread returns back to serve any other requests. Shared memory and semaphores are used to synchronize the message passing between the Java Client and the Sentinel application.

4.3 Application Testing

The Configuration Server has been used thoroughly and it proves to be an important tool in creating any factory configuration with any number of machines and batches and different parameter values. The user can either create a new configuration or delete an existing configuration, he can even view the existing configuration information.

The Virtual Factory Application has been tested for different projects created using the Configuration Server and the reports on machine status when requested by the user, give accurate results. The results are all shown as status reports. The batch flow can be seen from the front end.

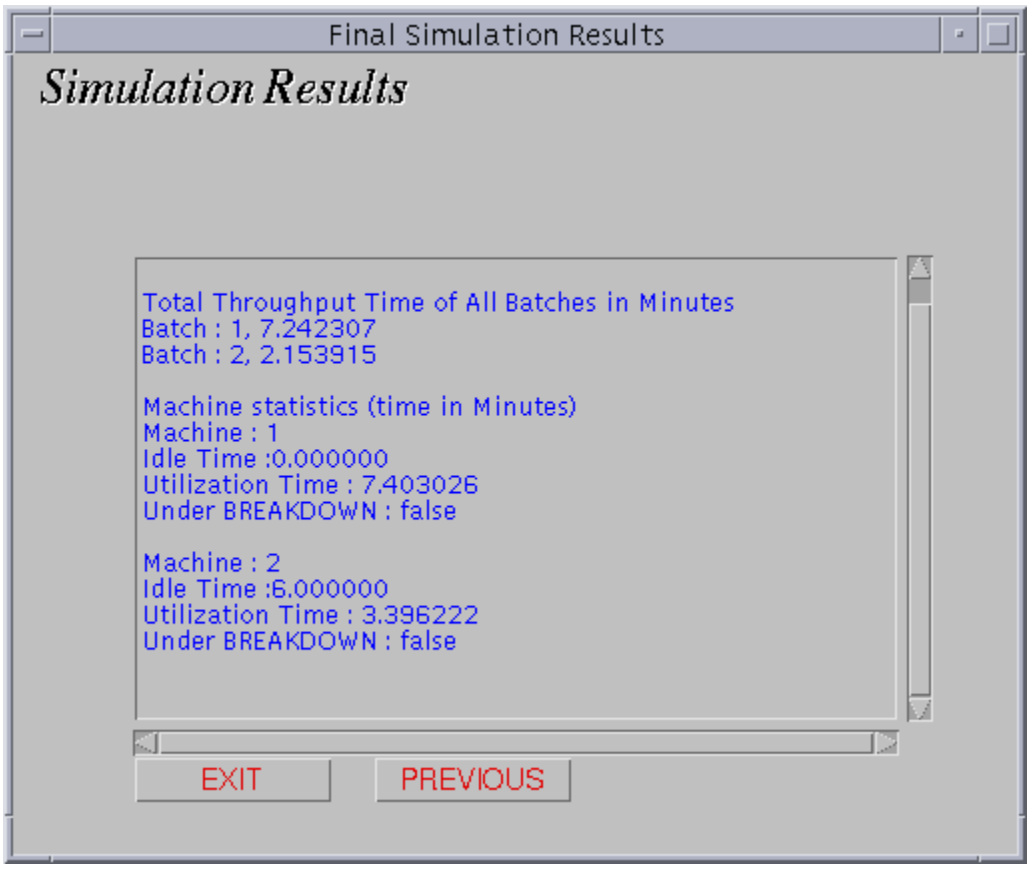


Figure 5.1 Simulation Results

The Final Simulation results are shown in Figure 5.1. The results include the total throughput time of all the batches, the idle time of individual machines, the total utilization time of the individual times and the machine breakdown condition.

In the above example, its shown that the first machine had no idle time and the second machine had relatively higher idle time due to the breakdown.

The other reports that could be generated are the individual machine reports that could be requested at any time of the simulation.

The machine reports for the two machines that were included in the simulation are shown in Figure 5.2 and Figure 5.3

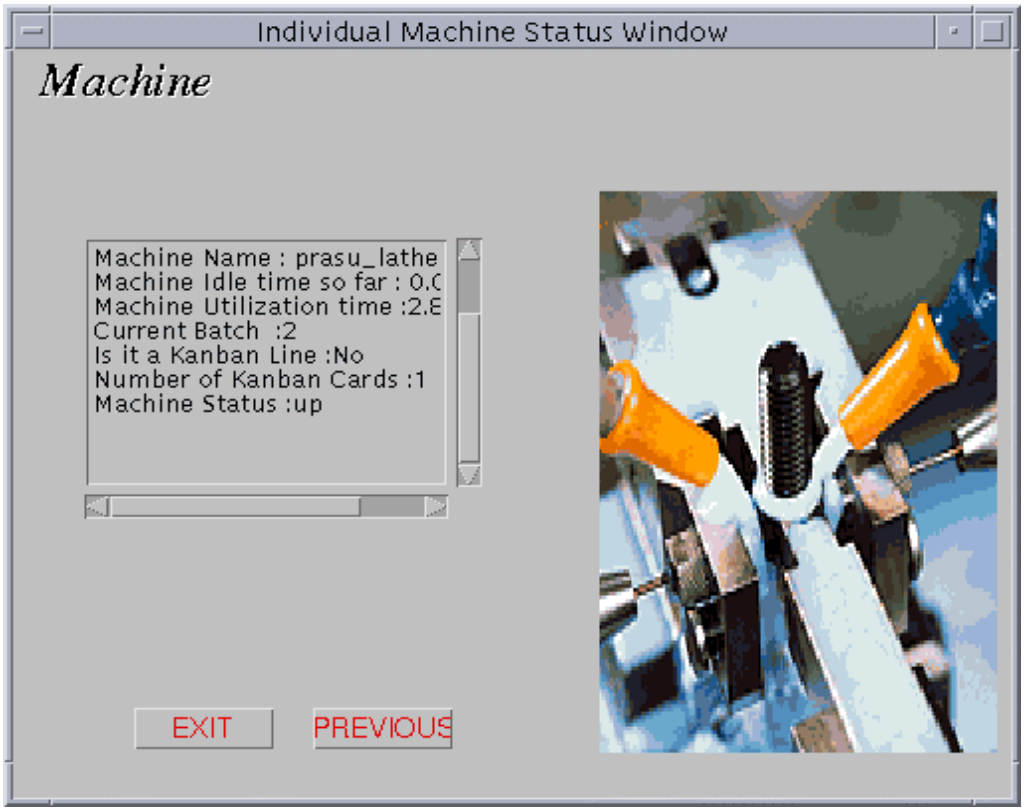


Figure 5.2 Status Report of Machine#1

The Machine Status reports show the machine statistics like the machine-name, machine-idle time, machine-utilization time, the current batch number that is being processed, if it is a Kanban production line, the number of Kanban cards(default value being 1) and the machine status. These statistics could be got at any instant of time.

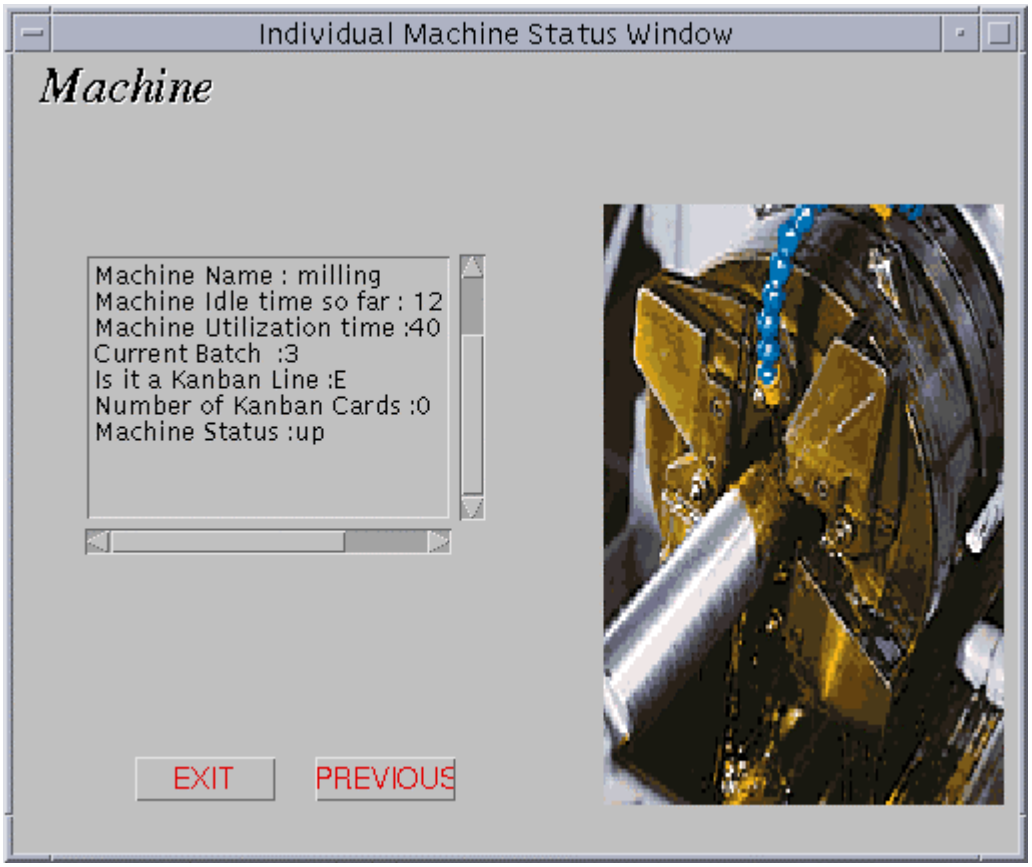


Figure 5.3 Status Report of Machine#2

Other than the status reports and final results, the simulation can be seen in an other frame created by the Java server side application All the various steps in the simulation process, including the repair events are shown by means of different labels here. This can be seen in Figure 5.4

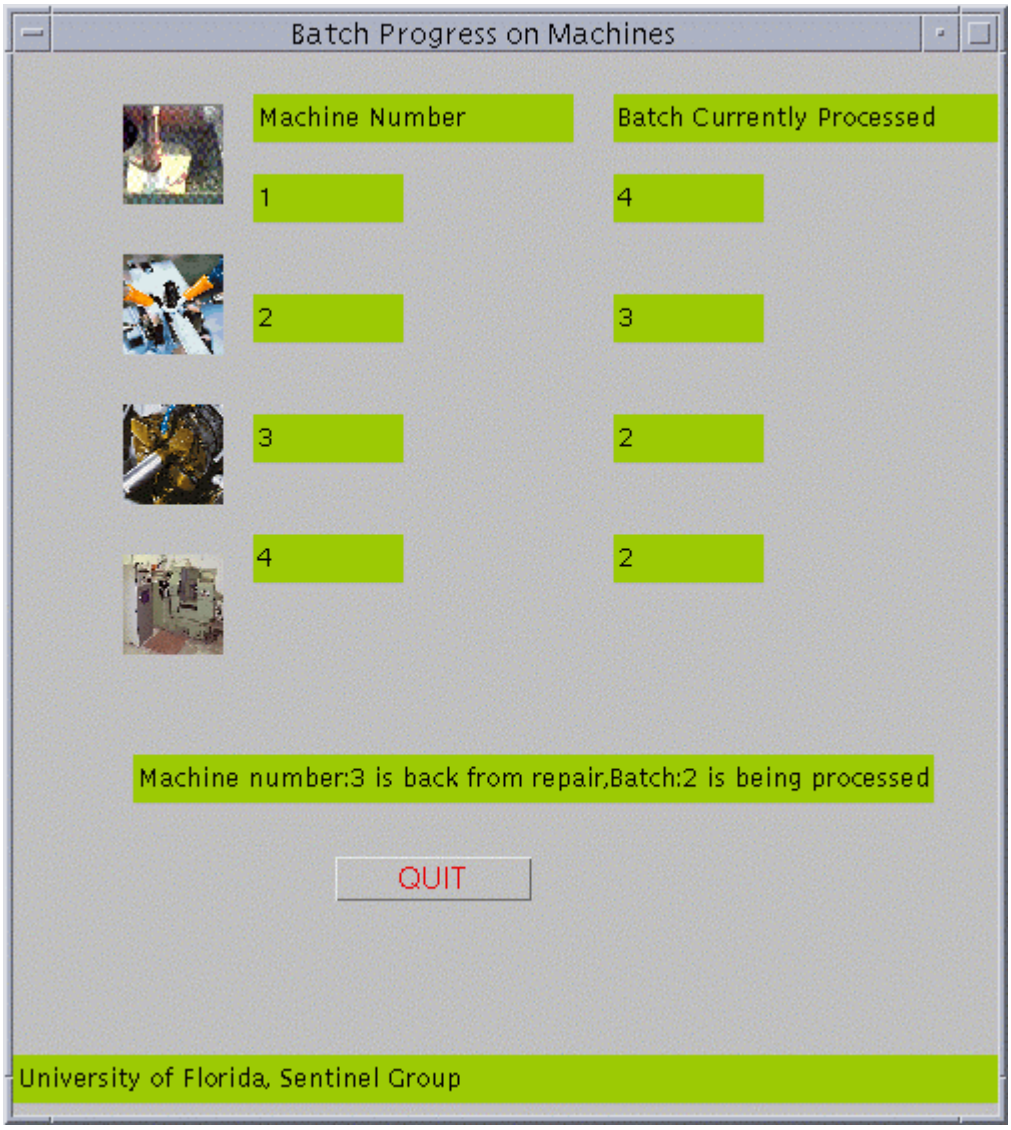


Figure 5.4 Batch Flow Simulation

The initial dialog box and the Project Selection frame can be seen in Figure 5.5 and Figure 5.6

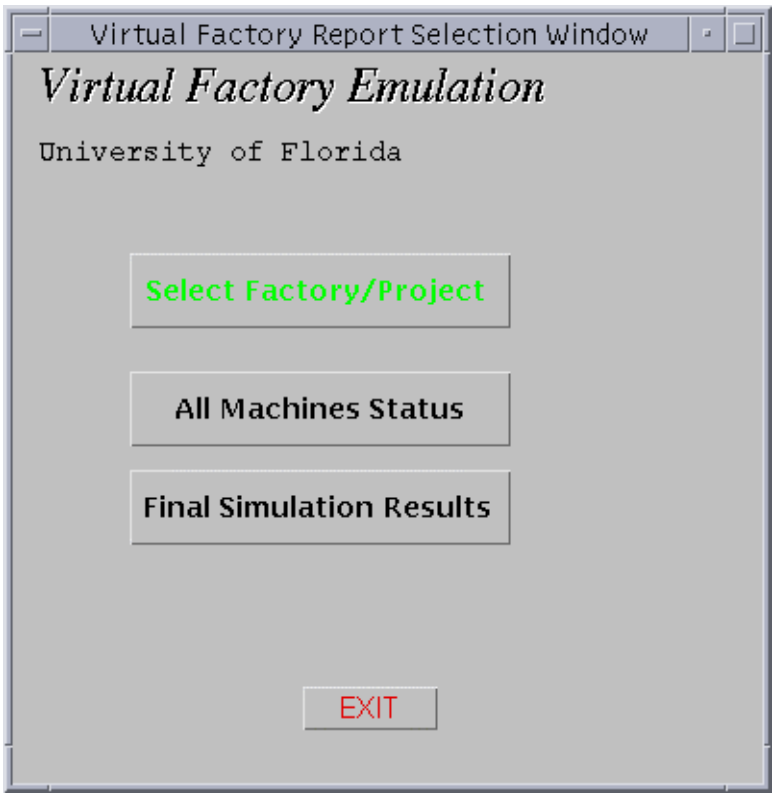


Figure 5.5 Initial Dialog Box

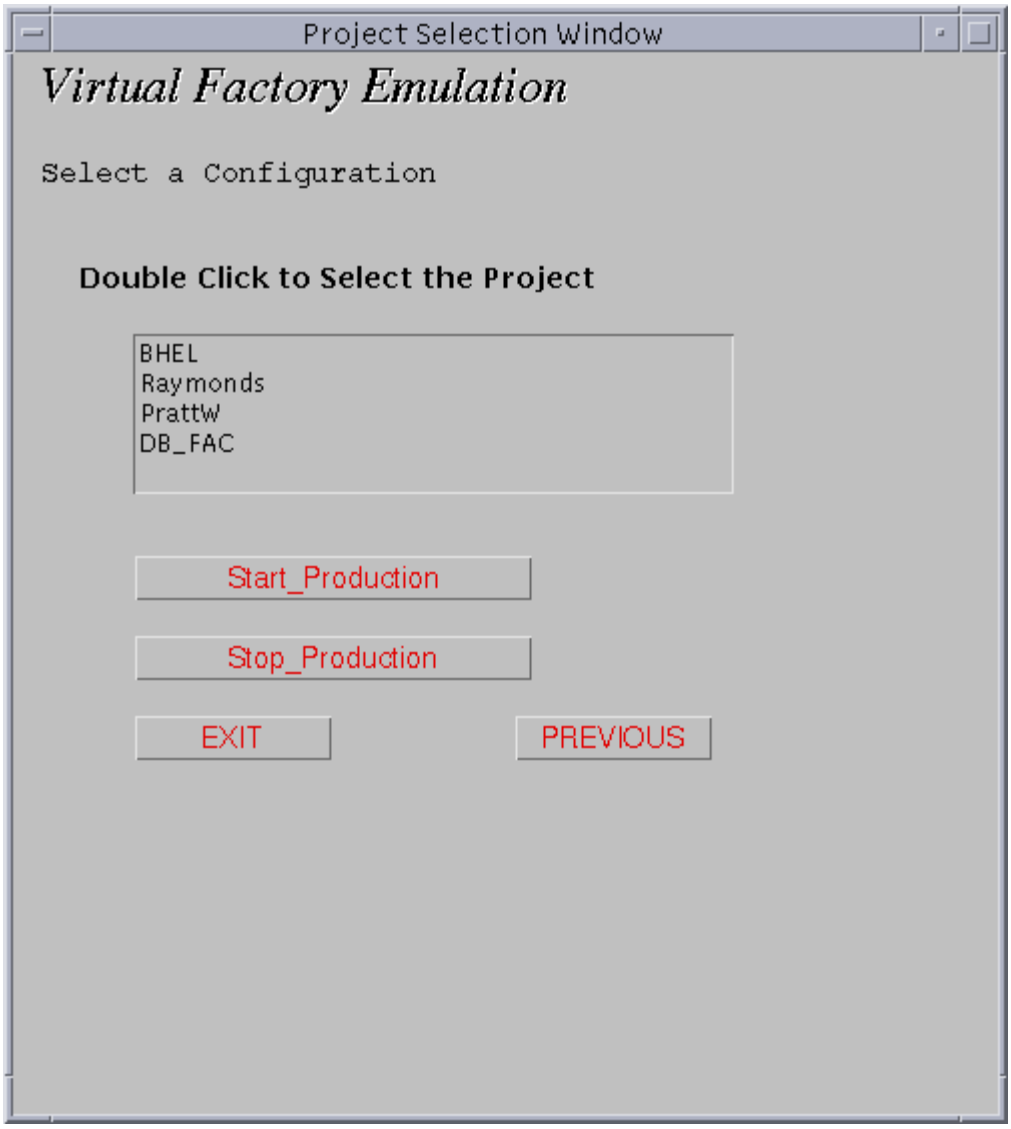


Figure 5.6 Project Selection Window

CHAPTER 5
CONCLUSIONS AND RECOMMENDATIONS

The purpose of this thesis was to evaluate the functionality of active object oriented databases like Sentinel for a real life application of a manufacturing unit or a production shop floor. Sentinel had most of the features that were required for this Virtual Factory Application. The features which could be applied for a real life application, and that have been tested here are:

- Sentinel can detect primitive events, composite events and complex events.
- It can detect the local events by means of a local event detector and global events by means of a global event detector. The complex systems could be divided into simpler, individual applications, which was made possible by GED.
- The specification of event-condition-action facilitates a better and simplified approach to event driven programming style.
- The environment variables and the application names can be set separately, and so the application developer need not make multiple copies of the same executable with different names.

Some of the recommendations for future work are:

- Other features and Manufacturing concepts like CONWIP and KANBAN can be extensively used and thereby this application could be developed into more of a useful tool for analysis .
- The Configuration Server and the Virtual Factory Implementation could be combined into one executable, so that the user can select a specific configuration and study the simulation at the same time.

APPENDIX A
DATABASES

Some of the commonly used definitions and terminology are explained in the following sections.

A.1 Database Definition

A database is a collection of inter-related, persistent data about one enterprise. The data could range from a collection of numbers, alphabets to images, or any combination of these.

A.2 Object Oriented Concepts

Sentinel is an integrated active OODBMS that supports Event-Condition-Action (ECA) rules and their management. The Open OODB Toolkit (from Texas Instruments, Dallas, Texas) is used as the underlying platform. Event and rule specifications are seamlessly incorporated into the C++ language. Any method of an object class is a potential primitive event. Furthermore, before- and after- variants of method invocation are allowed as events. Composite events are formed by applying a set of operators to primitive events and composite events. Typically events and rules are specified in a class definition.

In addition, Sentinel supports events and rules, which are applicable to a specific object instance. In that case, events and rules are specified outside class definitions within the program where the instance variables are declared. Supporting this instance-level events and rules allows composite events, which are combinations of events from different classes or different applications (global events).

The parameters of a primitive event correspond to the parameters of the method declared as the primitive event and other attributes, such as the time of occurrence. The processing of a composite event entails not only its detection, but also the computation of the parameters associated with it. The parameters of a composite event are collected, recorded and passed on to condition and action functions of a rule. Furthermore, these parameters are stored and recorded to satisfy parameter contexts that are used to capture the application semantics. Recent, chronicle, continuous, and cumulative parameter contexts (termed event consumption modes in the literature) are supported in Sentinel.

An event (primitive as well as composite) can trigger several rules, and rule actions may raise events that can trigger other rules. Sentinel supports multiple rule executions, nested rule executions as well as prioritized rule executions. The three coupling modes (immediate, deferred and detached) were introduced to support application needs. Currently immediate and deferred modes are supported in Sentinel.

A.3 Exodus Server

The Exodus Storage Manager, developed at the University of Wisconsin under DARPA sponsorship, is a multi-user object storage system supporting versioning, indexing, transactions, concurrency, and recover. Exodus is implemented in C and C++ and provides a C interface. Sentinel uses Exodus to provide persistent storage, transactions, concurrency control, recovery, and client-server communications.

A.4 SNOOP

Snoop is an event specification language used in Sentinel for specifying ECA rules. It uses an event hierarchy to classify events, defines the notion of events and event expressions, and describes a set of event operators for constructing composite events.

A.5.1 Events

Processing an event involves the following (in the temporal order in which they occur):

1. Event occurrence: When an event *occurs*, its formal parameters are instantiated (or bound) with the values of actual parame ters,
2. Event detection: When an event is *detected*, the parameters are collected and recorded by the event detector, and
3. Event signaling: the event is *signaled*, by sending an interrupt/message along with the actual parameters computed to the condition or rule evaluator.

A.5.2 Event Classification

On the basis of structure and behavior, events can be organized into a hierarchy of event classes. Each event class represents a unique event type (i.e. logical event) and instances of a class are identified by

their class type and time of occurrence. The term event is used to refer to both an event instance and an event class.

Events can be broadly classified into:

1. Primitive events – events that are pre-defined in the system (using primitive event expressions and event modifiers); a mechanism for its efficient detection is likely to be embedded in the system for each primitive event, and
2. Composite or complex evens – events that are formed by applying a set of operators to primitive and composite events.

Primitive events are further classified into:

(a) Database events: they correspond to database operations such as retrieve, insert, update and delete (in the relational model).

(b) Explicit events: they are detected and signaled along with their parameters by application programs and are only managed by the system.

(c) Temporal events: they are related to time and are of two types – absolute and relative. Absolute events map to discrete points along the time line, whereas relative events are defined with respect to an explicit reference point.

A.5.3 Expressiveness of Snoop

To provide an unambiguous way of composing event expressions in Snoop, there is a default precedence for event operators. The grammar for Snoop is shown below:

E :: begin-of E1 end of E1

E1::= E1 AND E2 | E1 OR E2 | E2

E2::= E2 SEQ E3 | E3

E3::= E3, E4 | E4

E4::= A (E1,E2,E3)

| A* (E1,E2,E3)

| P (E1, [time_string], E1)

| P (E1, [time_string]: parameter, E1)

| P* (E1, [time_string]: parameter, E1)

| [absolutetimestring]

| (E1) + [relativetimestring]

| Database Events

| E1

Value::= integer | ∞

Event expressions generated by the BNF are left associative. Parentheses are used for overriding precedence in a sub-expression.

A.5.4 Rules

A rule can have at most four options for detecting the event which it subscribes along with a few more parameters such as coupling mode, parameter context, event consumption mode, and priority of the rule.

Usage:

Rule r1[e1, check_price, set_price, RECENT, IMMEDIATE, NOW, 10];

Among the optional parameters of a rule definition, the parameter_context should be specified first, and others can be given in any order using the keyword match. The parameter_context of the event which a rule subscribes should be placed right after action_function if it exists. If several rules need to be defined on the same event in different parameter contexts, then the rule definition has to be duplicated for each context. Parameter_context can be one of the RECENT, CHRONICLE, CONTINUOUS, or CUMULATIVE.

Coupling mode defines the mode of the rule execution. Currently, IMMEDIATE and DEFERRED coupling modes are supported.

Priority classes are used for specifying rule priority. An arbitrary number of totally ordered priority class can be defined.

Rules can be specified at class definition time or as part of an application. Rule activation and deactivation are supported at run time. Named events can be reused later. Also, a number of rules may be defined on the same event expression.

An optional rule_trigger_mode for specifying the time at which event occurrences are to be considered for the rule. Two options, NOW (start detecting all constituent events starting from this time

instant) and PREVIOUS (all constituent events from past are acceptable) are supported as rule triggering modes, with NOW being the default.

A.5.4 Making Conditions and Actions

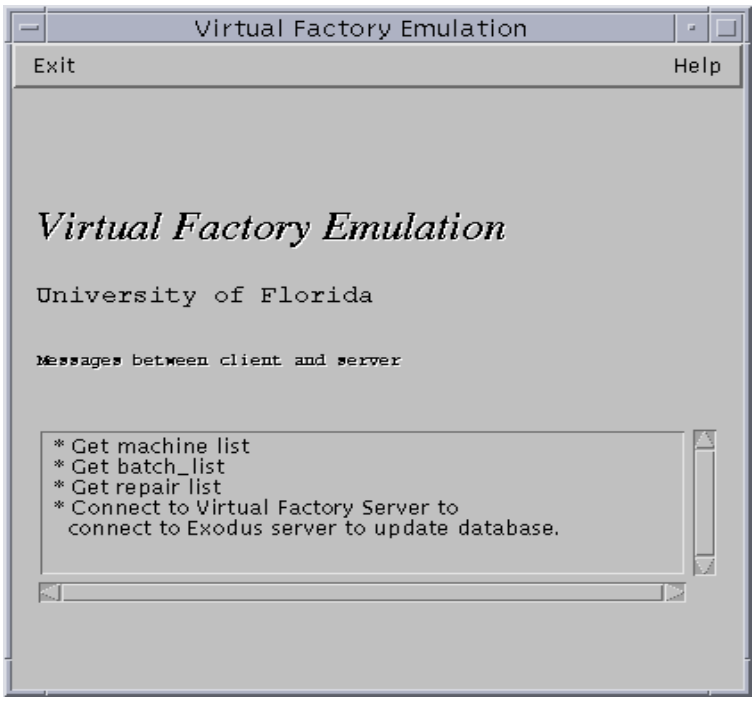
The rule specification involves specifying an event expression, a condition function (or an OQL expression), an action function, parameter contexts, a coupling-mode, priority and a rule-trigger mode. The rules can be specified either at the object level or in the main program. The Condition and Action functions are passed one parameter of type L_OF_L_LIST. This L_OF_L_LIST contains all the parameters of all constituent primitive events. Simple data types (e.g. integer, float, character and string by value) as passed as parameters.

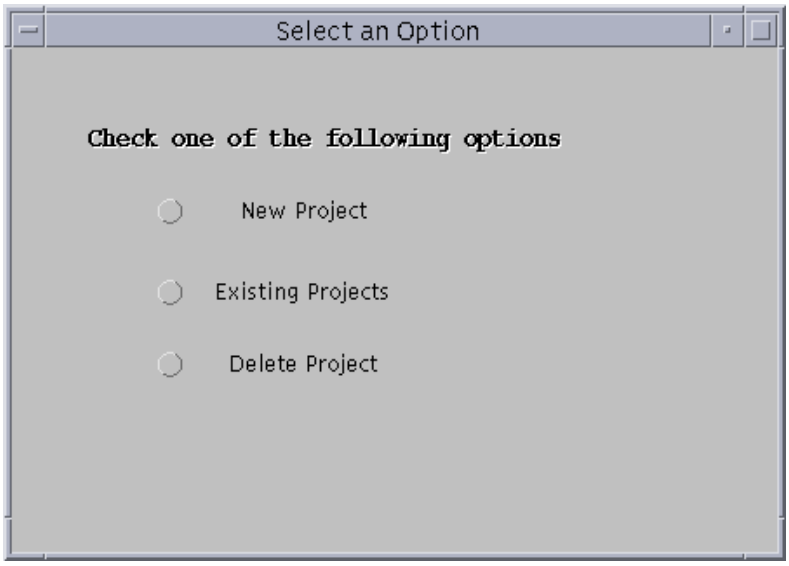
Example of Action Function:

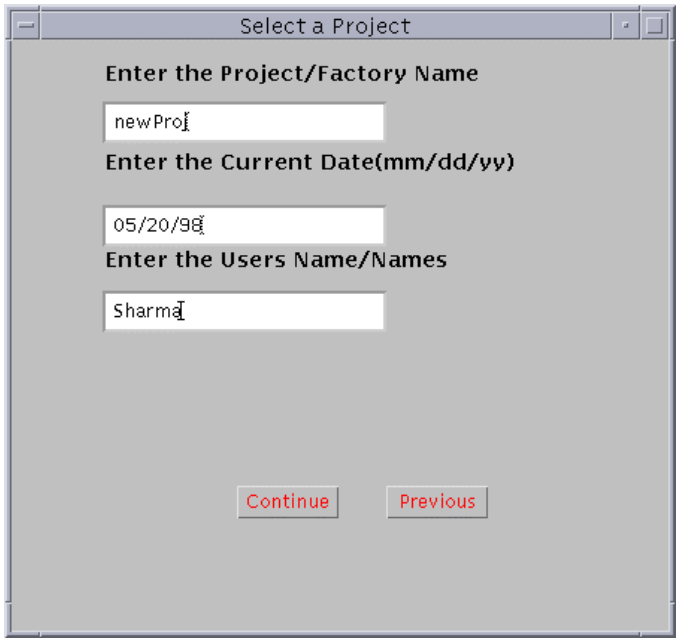
This action function sells 20 stocks of the Reactive object related to this event:

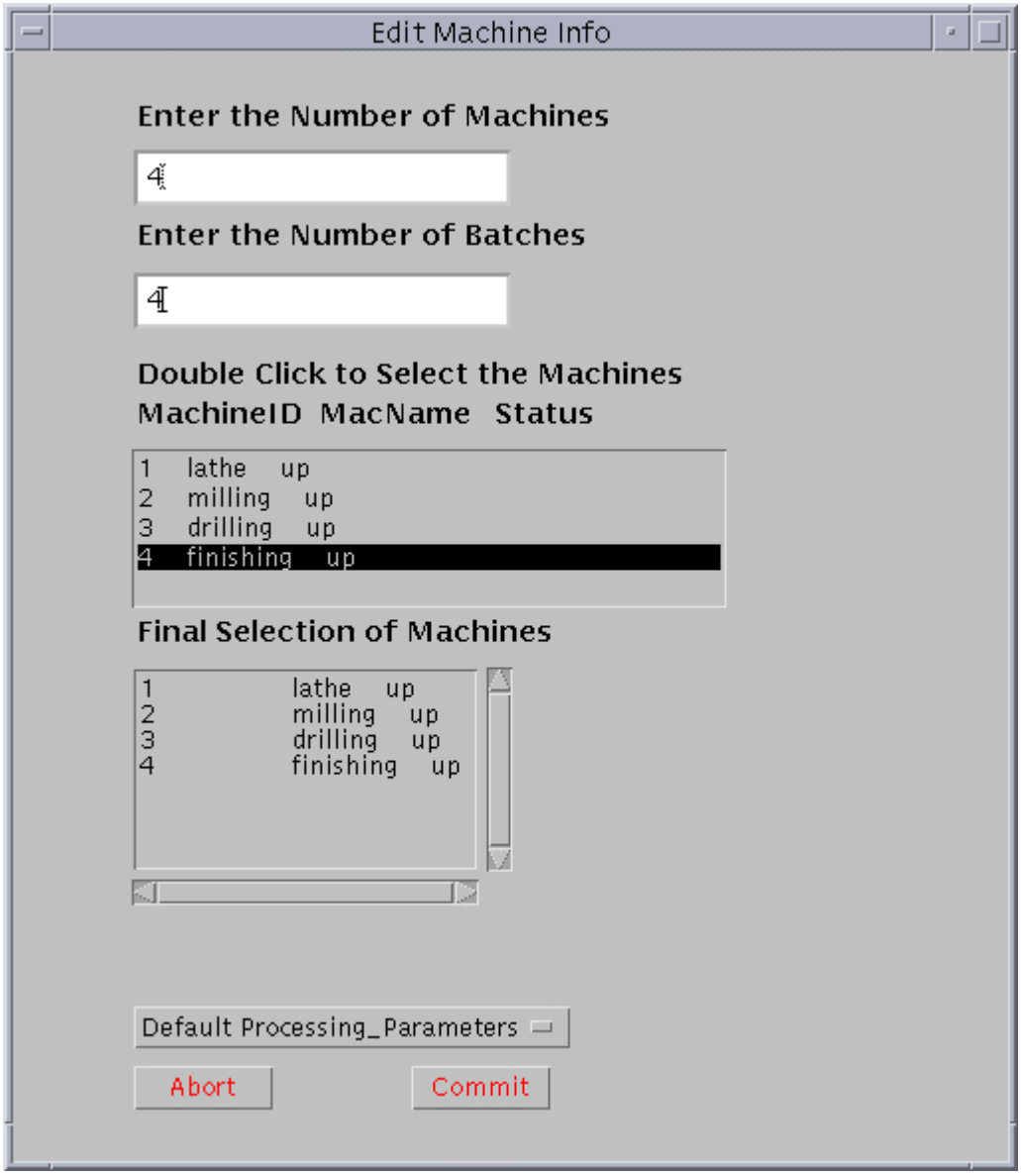
```
void action (L_OF_L_LIST)
{
        STOCK *u;
        LIST_OF_LIST *l1;
        LIST_NODE *l2;
        l1 = n1_list->getFirst();
        l2 = l1->get_head();
        u = (STOCK*)l2->get_reactive_obj();
        if (u->get_stock_bal > 20)
        u->sell_stock(20);
}
```

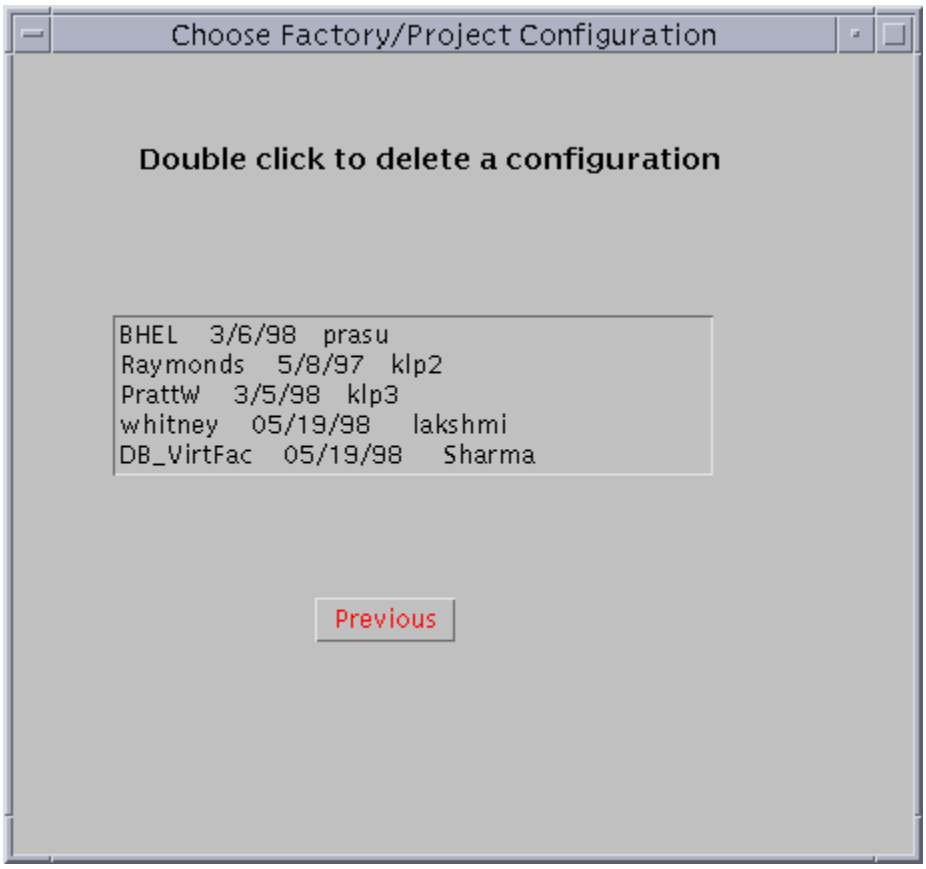
APPENDIX B
APPLICATION INTERFACE

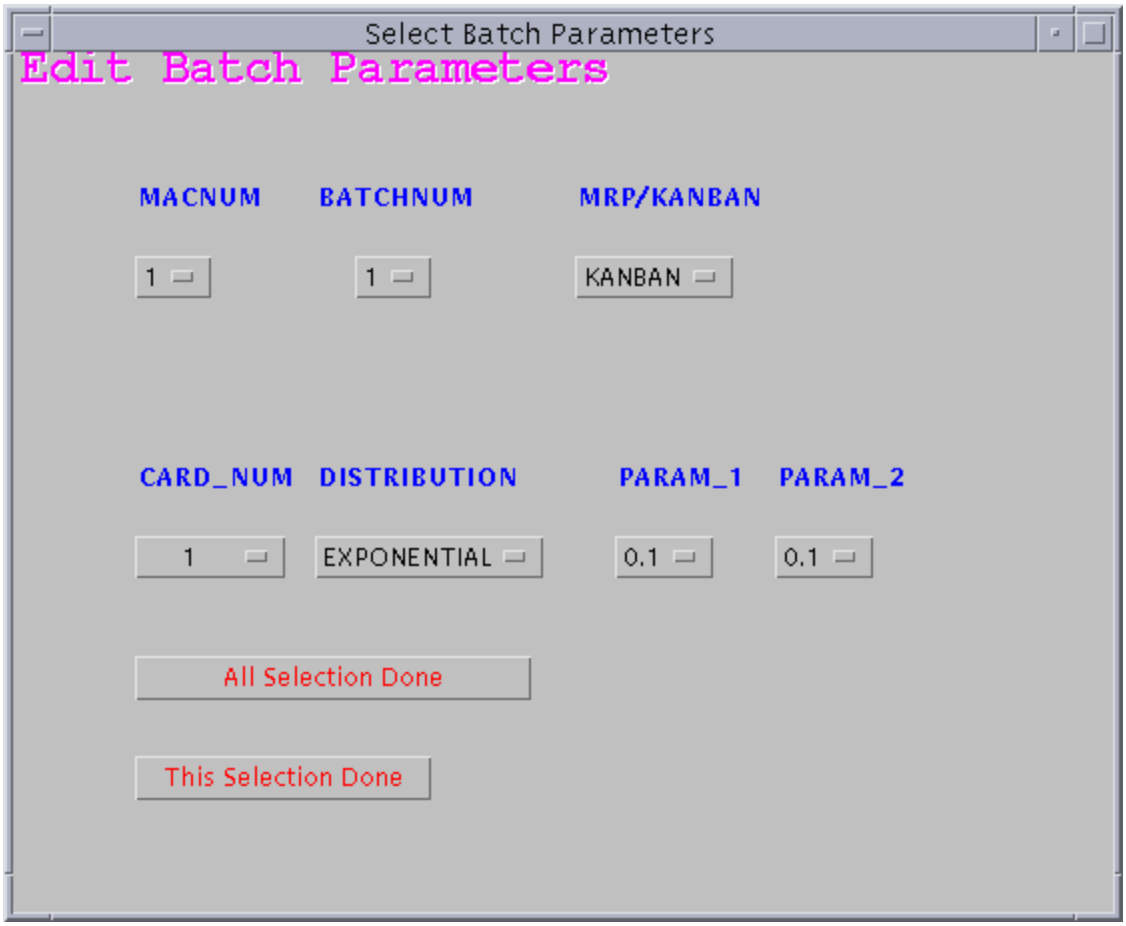



Select an Option
Check one of the following options
New Project
Existing Projects
Delete Project


Select a Project
Enter the Project/Factory Name
newProj
Enter the Current Date(mm/dd/yy)
05/20/98
Enter the Users Name/Names
Sharma
Continue
Previous


Edit Machine Info
Enter the Number of Machines
4
Enter the Number of Batches
4
Double Click to Select the Machines
MachineID MacName Status
1 lathe up
2 milling up
3 drilling up
4 finishing up
Final Selection of Machines
1 lathe up
2 milling up
3 drilling up
4 finishing up
Default Processing_Parameters
Abort
Commit


Choose Factory/Project Configuration
Double click to delete a configuration
BHEL 3/6/98 prasu
Raymonds 5/8/97 klp2
PrattW 3/5/98 klp3
whitney 05/19/98 lakshmi
DB_VirtFac 05/19/98 Sharma
Previous


Select Batch Parameters
Edit Batch Parameters
MACNUM
BATCHNUM
MRP/KANBAN
1
1
KANBAN
CARD_NUM
DISTRIBUTION
PARAM_1
PARAM_2
1
EXPONENTIAL
0.1
0.1
All Selection Done
This Selection Done

LIST OF REFERENCES

[1] Daniel Sipper and Robert L. Bulfin, Jr., Production: Planning, Control and Integration, McGraw-Hill, New York, NY, 1997.

[2] David Flanagan, Java in a Nutshell, 2ed, O'Reilly & Associates, Sebastopol, CA 1997.

[3] Debasis Mitra and Isi Mitrani, Analysis of a Kanban Discipline for Cell Coordination in Production Lines, Management Science, Vol. 36, No. 12, December 1990, pp. 1548-1566.

[4] Gabriele Elia and Giuseppe Menga, Edited by Sanjay B. Joshi and Jeffrey S. Smith, Object-Oriented Design of Flexible Manufacturing Systems, Computer Control of Flexible Manufacturing Systems: Research and Development, Chapman Hall, London, 1994.

[5] H. Liao, Global Events in Sentinel: Design and Implementation of a Global Event Detector, Master's thesis, Computer and Information Science Engineering, University of Florida, Gainesville, 1997.

[6] Hung-Ju Chu, A Flexible Dynamic ECA Rule Editor for Sentinel: Design and Implementation, Master's thesis, Computer and Information Science and Engineering, University of Florida, Gainesville, May 1998.

[7] J. Gosling and H. McGilton, The Java Language Environment: A White Paper. Sun Microsystems Inc., Mountain View, CA, 1995.

[8] R.K. Honnavalli, Design and Implementation of an Event-Based Shop Floor Control Application on Sentinel – An Active OODBMS, Master's thesis, Mechanical Engineering, University of Florida, Gainesville, 1995.

[9] Roberta S. Russel and Bernard W. Taylor III, Operations Management, 2ed, Prentice-Hall, Englewood Cliffs, NJ, 1998.

[10] S. Chakravarthy and D. Mishra, Snoop: An Expressive Event Specification Language for Active Databases, Data and Knowledge Science, Vol. 14, No. 1, 1994, pp 1-26.

[11] S. Chakravarthy, D. Mishra, S. Anwar and E. Maugis, Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules, Information and Software Technology, Vol. 36, No. 9, 1994, pp 559-568.

[12] S. Chakravarthy and Hyoungjin Kim, Sentinel User Manual for Sentinel 10.9, Computer and Information Science Engineering, University of Florida, Gainesville, September 1996.

[13] S. Chakravarthy, V. Krishnaprasad, Z. Tamizuddin, R.H. Badani, ECA Rule Integration into an OODBMS: Architecture and Implementation, Proceedings of the 11th International Conference on Data Science, 1995, pp 341-348.

[14] S. Manivannan and Jerry Banks, Design of a Knowledge-Based On-line Simulation System to Control a Manufacturing Shop Floor, IIE Transactions, Vol. 24, No. 3, July, 1992, pp. 72-83.

BIOGRAPHICAL SKETCH

Lakshmi Prasanna Kuppala was born in Hyderabad, A.P., India. She received the Bachelor of Engineering, Honors degree in Mechanical Engineering from Birla Institute of Technology and Science, Pilani, India, in May 1996. She will receive her Master of Science degree in Mechanical Engineering from the University of Florida, Gainesville, in August 1998. Her research interests include CIM Databases, CAD/CAM and Designs.