Design and Implementation of The MIDAS-2 Manufacturing Database System

A Project Report

Submitted in partial fulfilment of the

requirements for the Degree of

Master of Engineering

in

Computer Science and Engineering

by

N. Sridevi

Department of Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012

January 1997

Contents

Acknowledgments

I would like to thank my project guide, Dr. Jayant Haritsa for his excellent guidance and inspiration. I am grateful to him for his understanding and readiness to help me any time I approached him. He took serious efforts to provide a very good working environment in Database Systems Lab and all the facilities needed. I also thank Dr. Y. Narahari for his suggestions and help in every step of my project.

Bhuvana has helped me a great deal in the initial stages of the project. I thank her for the time she spent in giving me an insight into the MIDAS project. I thank my lab mates and friends Shrividya, Narayanan, Mouli, Ramesh and Binto for their help and for making my work enjoyable. Special thanks to Nalin for his valuable suggestions.

I would like to thank my close friends in IISc and elsewhere for their encouragement. Finally, I would like to thank my parents for their love and patience.

Abstract

Flexible Manufacturing Systems (FMS) have evolved as an integrated approach to automating not only the production operation but also the support, planning and scheduling functions. Commercial database systems are based on the relational model and cater to business applications, but they are not suitable for handling the complexities and unique features encountered in the manufacturing environment. To address this lacuna, the MIDAS (Manufacturing Information DAtabase System) project was started at DSL (Database Systems Lab), Indian Institute of Science. The goal of this project was to design and prototype a new database system for effectively supporting the management of flexible manufacturing systems. This database is based on the object-oriented model and incorporates active, temporal and real-time features. An initial prototype (MIDAS-1) was developed in an earlier work that dealt with the issues of classifying and analyzing the data in the manufacturing domain, developing the object model, integrating active mechanisms and providing simple decision support. The basic MIDAS system was customized for a Coffee Manufacturing Plant example.

In this project, we overcome some of the limitations of the initial system by adding advanced features and call this enhanced system as MIDAS-2. First, the control aspects of a manufacturing plant are captured in the dynamic model designed based on the Object Modeling Technique (OMT). Second, the decision support system of MIDAS-1 is augmented with a more effective scheduling technique based on Lagrangian Relaxation. This scheduler tries to minimize the tardiness of the jobs and the work-in-process inventory. Third, real-time response to the plant activities is supported by incorporating a Real-Time Data Server. The server acquires temporal data sampled from the plant, serves it to requesting clients and saves it in the database in real-time.

The MIDAS-2 system is customized for a semiconductor device manufacturing plant example that is a representative example of an FMS with flexible automated machines, complicated routing and a variety of products. The MIDAS database is used to incorporate some active mechanisms to aid in monitoring of the semiconductor plant. Finally, a Multimedia User Interface was provided for this plant, which reflects the latest state of the plant through a graphical interface and informs the operator of exceptional events through an audio interface.

Chapter 1

Introduction

1.1 Overview

Flexible Manufacturing Systems (FMS) have evolved as an integrated approach to automating not only the production operation but also the support, planning and scheduling functions. FMS is characterized by its ability to respond quickly to product mix and design changes through the use of flexible automated equipment. The FMS system must have access to current information about the plant organization and operation in order to be able to adapt to changes in the environment. Thus, it requires to communicate with the various sub-systems in the manufacturing activity like product design, production planning, shop floor management and production analysis. The various manufacturing processes can be integrated through a well-structured database that provides an interface between all the functions of the FMS.

The data in the manufacturing domain is heterogeneous and complex. A classification of the this data is given below.

- Structural Data

  The structural data describes the physical and logical construction of the plant. Structural data in a manufacturing plant comprises of information about the plant layout, customer details, part programs for numerically controlled machines, configuration details of machines and controllers, inventory details, product details, production details such as lot descriptions, etc. This data is "static" in the sense that it changes very slowly, for example when a new machine is added to the shop floor or when a new part program is designed.

- Status Data or Temporal Data

  The sensor (or measurement) data is the raw information received from the plant's monitors and sensors. Sensors could be measuring continuously changing values such as the position of a moving AGV, or they may be supplying data about an event such as a part detected to be defective and workpiece completing a processing step. A *history* of sensor data may have to be maintained to be used later in performance and fault analysis. Status data forms the bulk of manufacturing data due to its high arrival rate. Thus, periodical archiving or versioning at coarser granularity can be done to deal with the storage requirements.

- Performance Data

  The status information collected from the plant can be used for analysis of the plant's operations. Performance parameters such as throughput, waiting times, transportation times, machine utilizations, and workshop loads derived from the status data form the basis for the evaluation of the plant.

- Control Data

  The current settings of the machines and controllers in the plant form the control data. In addition, there may be a library of the pre-defined control settings that can be used for a variety of common manufacturing operations.

The volume of data generated is in the order of gigabytes per day. Hence, the magnitude and variety of data calls for efficient data management and information retrieval.

Commercial database systems are based on the relational model and cater to business applications, but they are not suitable for handling the complexities and unique features encountered in the manufacturing environment. To address this lacuna the Manufacturing Information DAtabase System (MIDAS) project was started. This project aims to develop a database system that deals with the control and data management issues of flexible manufacturing systems.

The initial prototype of MIDAS was developed in an earlier work, which we refer to in this thesis as MIDAS-1. The various entities of the manufacturing domain were identified and modeled, and the MIDAS architecture was designed. The MIDAS System was built using the Object Methodology Technique (OMT) [Rumb+91]. The data model was implemented and the system was tested for a simple automated environment. The detailed description of MIDAS-1 is given in [Bhuv96].

In our work, which we refer to as MIDAS-2, we have added some advanced features to overcome some of the limitations of MIDAS-1. Decision support for scheduling the manufacture of products with bills of materials, and support for Real-Time acquisition and service of temporal data were incorporated in MIDAS-2. In addition, the dynamic aspects of the manufacturing domain were modeled. Finally the total system was tested for an automated plant example.

1.2 The MIDAS system

The goal of the MIDAS system is to design and prototype a new database system for effectively supporting the management of flexible manufacturing systems. The principle of MIDAS system is to have the factory operator interact solely with the database, that is, the database logically embodies the manufacturing system. A database system for FMS has architectural requirements, temporal requirements, automation requirements and performance goals. The MIDAS database is designed to provide all these complex functionalities in an integrated fashion. Thus, the MIDAS design has to support complex objects, active mechanisms, and embedded control, all of which are essential to the manufacturing domain. The development of the MIDAS system has taken place in two phases. In the first phase an initial prototype (MIDAS-1) was developed [Bhuv96] and some advanced features were added during the second phase (MIDAS-2), which is described in this thesis.

1.2.1 MIDAS-1

Object oriented databases appeared to be most suited to handle the type of data encountered in manufacturing systems. Initially, the manufacturing domain was studied and the objects of this domain were identified. Since the object-oriented approach helps to represent the real world objects naturally, it was used to model the manufacturing domain. Subsequently, the architecture of the system was designed. The model was carefully analyzed and adapted to a database schema. The implementation of the data model used the Illustra object-relational DBMS as the backend engine [IUG95]. The implementation included a heuristic scheduler proposed in [DJ90], a view generator for displaying the plant status and other monitoring information, and a simplified control panel for operator intervention. The view generation programs are generic and not specific to a plant and they interact only with the MIDAS database to retrieve information. The MIDAS-1 system also included routines to access the database and control routines to handle simple events. The prototype was tested through the example of a Coffee Making Plant. This example though a simplified version of an FMS, helped to illustrate the principle of MIDAS design.

While the above prototype dealt with the data modeling issues and provided basic decision support, it has some limitations.

- The object model which was developed gives the *static* structure of the objects in a system and their relationships. It does not reflect how the control flows through the system, and how the various entities in the manufacturing domain react to the events occurring in the manufacturing plant. Thus, there is a need to model the control or *dynamic* aspects of the system.

- A heuristic scheduler based on the algorithm suggested in [DJ90] is used to generate a schedule for processing of parts in the FMS. The algorithm orders the set of jobs competing for each machine based on the job slacks and critical factors of each machine. The performance of the schedule generated from this scheduler is not quantifiable, since it does not give any bounds on the level of suboptimality of the schedule. Scheduling algorithms which use mathematical techniques should also be incorporated since they have predictable performance. These algorithms are much slower than the heuristic methods but produce schedules whose cost is only slightly above the lowest cost attainable (*i.e near-optimal*). The tradeoffs are between the quality of the schedule and the complexity of the algorithm. The user can choose the scheduler most suited for his manufacturing environment.

- We have seen in the previous section that status data like the state of a machine, job status, and lot status varies relatively slowly compared to the values of the sensors and monitors, position of material handler, etc., which change at a very high rate. The temporal data that changes very frequently needs to be sampled at a certain rate and stored along with the time stamp. This data will be retrieved for fault analysis and for immediate display of current status to the plant operators. Thus there is a need to provide the capability for *real-time* storage and retrieval of temporal data.

We address the above limitations in MIDAS-2. In the next subsection, we describe the additional features incorporated in MIDAS-2.

1.2.2 MIDAS-2

The features incorporated in MIDAS-2 are the following:

Dynamic Modeling

The MIDAS system utilizes features of the backend database engine to provide embedded control. In order to design the required control routines, a clear understanding of the dynamic behavior of the objects in the manufacturing domain is required. The Object Modeling Technique proposed in [Rumb+91] suggests developing a dynamic model of the system for this purpose. The Dynamic Model reflects the flow of control, interactions, and sequencing of operations in a system of concurrently active objects [Rumb+91]. The model gives the change of *state* of every object in response to external *events*. Examples of events that can occur in the manufacturing plant during production are machine breakdown, end of processing, request for part program by the processing unit, and dispatch of a transporter. In response to these events the object may in turn generate another set of events. Thus the sequence of operations that occur in response to the occurrence of an event can be traced. We have identified the major events that occur in the manufacturing process and their impact on each of the objects in the form of a *state diagram*.

Decision Support

One of the steps in planning the operation of production is production planning which determines the quantity of products to be manufactured during a particular period of time [VN92]. The decision is based on the available resources such as machine capacities, raw material and tools that are available or can be acquired. Next the scheduler tries to prepare a static schedule that has good on-time performance and which reduces the work-in-process (WIP) inventory.

We have implemented a scheduler based on the *Lagrangian Relaxation Technique* which has been shown to produce near-optimal schedules [CL94]. The performance of the resulting schedule can be quantified. It produces a single complete schedule for products with “bills of materials”. A bill of materials gives the raw material requirements of a product and specifies the order in which various parts have to be processed or assembled. The scheduler can be made to reconfigure an existing schedule to accommodate dynamic changes, by using the job interaction information that was generated while preparing the original schedule [Hoit+90]. This will eliminate the need for regenerating the schedule.

Real-Time Data Server

We have developed a real-time data server based on the design given in [Shim+93], which periodically acquires temporal data from the plant and transmits them to clients who request for the data. The clients could be graphical software tools that present the status data to the plant operator in an easily understandable form.

The real-time server consists of multiple threads scheduled based on *rate monotonic* *algorithm* proposed by Liu and Layland [LL73]. As mentioned before, the two main tasks of the data server are data acquisition and service. It has to acquire the temporal data in strict periodic manner and store it with a time stamp. This is a hard real-time task, that is, in no case should the system fail to meet the deadline, since we have to guarantee a certain accuracy of the temporal data stored. On the other hand the periodic transfer of this data to the clients is a firm real-time task. A firm real-time task tries to maximize the number of jobs that meet their deadlines, but if a deadline is crossed the job is discontinued. Past temporal data in arbitrary periods can also be requested for performance analysis or fault analysis. These requests do not have deadlines.

The server is a multi-threaded object that performs the functions of acquiring data from the plant, saving the data in permanent storage, receiving requests from clients, sending acknowledgments, and servicing the periodic and aperiodic data requests. The server is interfaced to the plant from which it acquires temporal data, and to the database where it stores the data.

Adapting MIDAS to an Example Plant

The total system was tested for a simplified semiconductor manufacturing plant. The object model encompassed all the entities in the new application. The semiconductor plant example has a variety of products, complicated routing for products, flexible automated machines and material handlers, and sensors and defect detectors that makes it a suitable example for our purpose.

Multimedia Interface

A graphical interface is provided for the new application that is intended for the operators viewing the plant status. The status of the plant, as indicated by the current states of the machines, transporters and sensors is reflected in the interface. An audio interface is provided to notify the plant operator of exceptional events. The view generation programs developed in MIDAS-1 are incorporated into the new application interface.

1.3 Related Work

The book [VN92] gives a comprehensive overview of automated manufacturing systems and a detailed description of flexible manufacturing systems. An introduction to computer automated manufacturing is given in [Powe87]. The book [Lugg91] gives a general description of flexible manufacturing cells and systems, and information about FMS implementation and installation.

Most manufacturing systems use the relational model for their factory databases. For developing an integrated database, the object model is more suitable in representing the various kinds of data encountered in the manufacturing domain. Object-oriented data modeling techniques are described in [Booc94, Rumb+91]. ROSE, a database management system for interactive engineering applications, exploits the superiority of the rich data model that the object paradigm offers and supports an object oriented interface for representing the designs of the engineering objects [HS89]. The generic data model for the integrated information system for the Hot Strip Mill, TISCO uses the inheritance and encapsulation features of the object paradigm to generalize a product family [KR95]. Both the systems referred to above use a relational DBMS for data storage and management.

A real-time data server for acquiring and servicing temporal data from the plant is proposed in [Shim+93]. The real-time server in MIDAS-2 uses this design. This server was also used in [Taka+96] for feeding a view generator that produced comprehensive views of the plant. The authors describe the *plant, process* and *lot* views, which are useful for monitoring purposes. MIDAS-1 provided views on these lines.

1.4 Organization of the report

The rest of the report is organized as follows. Chapter 2 gives a brief description of FMS and Chapter 3 provides an overview of the work done as part of MIDAS-1 and gives the architecture of MIDAS-2. Chapters 4, 5 and 6 describe the advanced features included in MIDAS-2 relating to dynamic modeling, decision support and real-time requirements respectively. Chapter 7 describes the semiconductor manufacturing plant example and the adaptation of MIDAS-2 to this plant. Chapter 8 gives the details of the Graphical Interface provided for the example plant. Finally, we summarize our work in Chapter 9.

Chapter 2

The Manufacturing Domain

This chapter gives a brief description of the various aspects of flexible manufacturing systems. We start with an introduction to CIM, and then move on to describe the organization and control hierarchy of FMS. The next part of the chapter describes the types of manufacturing data and the issues concerning its retrieval. The description presented in this chapter is a summary of the details given in [VN92, Lugg91, Rhod85, Powe87].

2.1 Computer Integrated Manufacturing

Computers have been appearing in different parts of manufacturing operations ranging from product design, production planning, and production control to business activities like marketing, order entry, and customer billing. Computer integrated manufacturing aims to integrate all these computerized operations to make manufacturing management control and decision making easier [Sepe87].

The manufacturing subsystems used in automating the various stages of manufacturing are listed in this section [Powe87, VN92, Sepe87]. The Computer Aided Design (CAD) system uses computer graphics for automating design of products and tools. The Computer Aided Engineering (CAE) system is a tool for engineering a product or system before it reaches the design stage. Computer Aided Process Planning (CAPP) is used to derive manufacturing routings for the product from engineering design data. The Material Resources Planning (MRP) system converts the master production schedule into a detailed schedule for purchasing raw materials and for manufacturing subassemblies and components. The schedules are based on manufacturing status, capacity as well as planned requirements. The Manufacturing Resource Planning (MRP II) ties in the resource and financial implications of production decisions. MRP is a subset of MRP II. MRP II system is a planning tool for handling the flow of material and work through the manufacturing process, inventory control, material data billing, cost development, capacity requirements planning, and purchasing. The Shop Floor Control (SFC) system converts the planning decisions into production orders. It releases detailed orders for production, schedules the orders, controls the production processes, and monitors the work-in-progress and factory status. The FMS is a sophisticated SFC system which in coordination with these software tools provides automated and controlled production of a variety of products.

2.2 Flexible Manufacturing Systems

FMS is an integrated approach to automating a production operation. An FMS is a computer-controlled manufacturing system that ties together automated production machines and material handling equipment, which is designed to adapt quickly and effectively to changes in product mix, demand, and designs [Powe87, VN92]. The subsystems are versatile computer-controlled equipment such as numerically-controlled machines, Automated guided vehicles (AGVs), coordinated measuring machines, and robots. The FMS control system controls the flow of parts through the system, making scheduling and dispatching decisions and responding to various planned and unanticipated events in real-time. In the following subsections we present a discussion on FMS [VN92].

2.2.1 FMS Organizational Structure

The FMS consists of a set of manufacturing cells which are dedicated to perform a set of related operations for a family of parts. This cell organization is based on group technology. Group technology is a manufacturing philosophy in which similar parts are identified and grouped to take advantage of their similarities in manufacturing and design. Each cell comprises of one or more automated machines and material handlers which cater to part movement within the cell. The FMS system structure comprises of versatile production machines with minimal changeover times between part types, versatile material handling system for timely movement of parts and tools, and on-line computer control of parts, tools and storage [VN92]. In the next subsection we discuss the control system of the FMS, which is responsible for scheduling of parts for processing, dispatching of machines and material handlers for operations, tool management, system monitoring and diagnostics, and reacting to disruptions to normal operations.

2.2.2 Control hierarchy

The control system is typically an interconnected network of the programmable controllers, cell controllers and the supervisory computers. The software resident in the various controllers has the capability to enable automated operation as well as monitoring and diagnostics.

The entire control hierarchy of a full fledged FMS has a depth of four levels [Rhod85]. This is shown in Figure 2.1. The supervisory controller or the corporate level controller coordinates the different manufacturing activities. It manages the long term planning activities such as master production planning. Under the supervisory controller lie the MRP II, CAPP, CAD tools and the factory level controller.

The Factory Level Controller or the Factory Control System (FCS) coordinates and controls the cell controllers. It manages the scheduling of material handlers across cells and the assignment of jobs to each cell. It handles the routing of workpieces through the cells from the loading station to the unloading station. It has direct access to the DBMS, which has the part programs, tool databases, routings, and equipment status information.

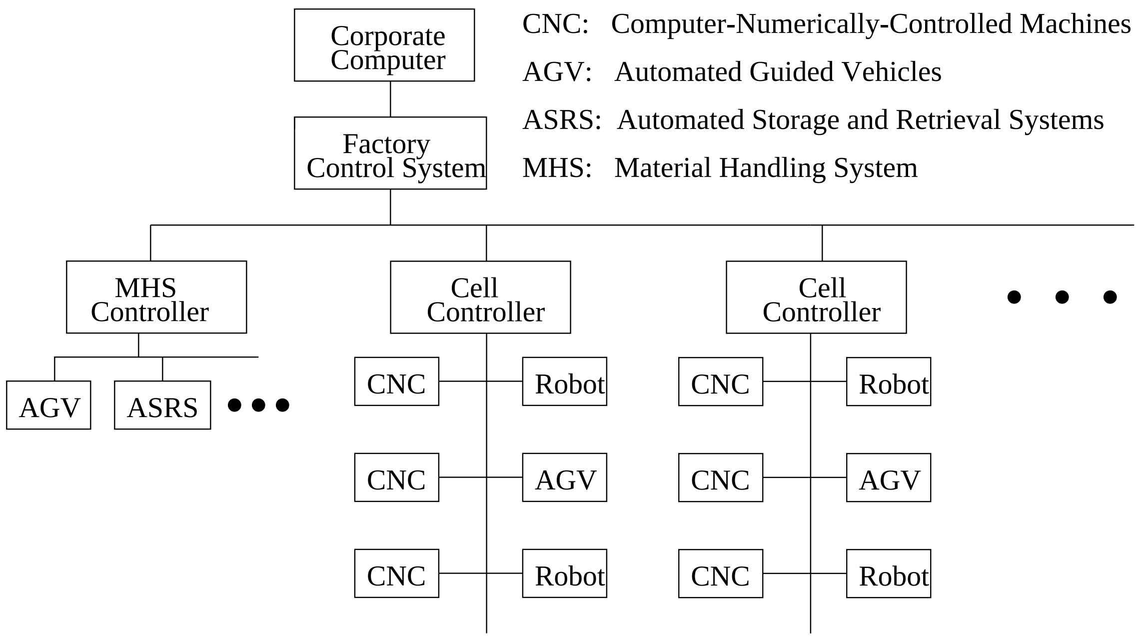


Figure 2.1: Control hierarchy for an FMS

The system has also incorporated into it expert systems to handle activities such as short term scheduling, part routing, deadlock resolution for transporters and decision support to handle disruptions and unanticipated events.

The Cell Level Controller handles the control of machines and the routing of workpieces through the machines within the cell. It dispatches the machines for the various operations and sequences the jobs for service by the material handlers. It takes decisions regarding which part is to be processed next on a machine, and which job to be serviced by a material handler when it is ready. It has the capacity to detect failures in equipment in the cell and take corrective action. Error conditions are signaled by intelligent resources such as expert system or by sensors. In the situation where a distributed database is used, some of the functions of the supervisory controller, like, retrieving part programs, routing of part types, tool databases, etc. are done by the cell controller.

The controllers of the automated equipment like CNCs, AGVs, ASRSs, and Robots are called the equipment controllers. The equipment controller handles control and monitoring of the equipment. It interprets the part programs transferred by the cell controller and configures the equipment for a particular type of operation. It also monitors the operation of the equipment and sends status information to the cell controller. Some CNC machine controllers may have probes for broken tool detection, part misalignment, etc., and can stop the machine operation in case of disruptions.

In the hierarchical control structure described above, control information flows from the corporate planning to shop floor controllers and status information flows upwards. The details of the information flow are given in the next subsection.

2.2.3 Controller Communications

A variety of messages are passed between the different controllers. The part programs and operational data are stored in a centralized database accessible via the FCS or distributed database at each of the cell controllers. They are fetched from the database on request and downloaded to the equipment through the FCS or the cell controller. Temporal information about the part type being worked on, number scheduled, number waiting, and equipment status are communicated to the cell controller. Production information such as throughput, waiting times, transportation times and machine utilizations are also conveyed to the cell controller. Information regarding the status of equipment, tool usages, fixtures, raw and semi-finished workpieces are stored in the database. Real time data for time-critical operations such as machine feedback, quality control data, monitoring and diagnosis information, AGV routing, and part flow data are also transferred to the supervising controller, which enables it to react to anticipated and unanticipated events with minimum delay. The status information is stored in the database at the cell controller in case of distributed environment and passed on to the factory level database if it is centralized.

2.2.4 Database Management System for an FMS

The integration of product design and manufacturing processes through a well-structured database is a key to Computer Integrated Manufacturing. The CAD database contains the design data for products. The MRP database contains data required for planning production which is the statistics of the various resources available and of the demand for the various products. The financial and administrative information is stored in a separate database. The manufacturing process related data is stored within the FMS. This database could be centrally located with the factory level controller or could be distributed. Most automated plants use a distributed database. In the distributed case, typically, general information about the plant, the products, plant configuration details, schedules, software for expert systems, MHS, etc. are present in the factory level database. The cell controllers in this case have their own database containing the part programs for the automated machines in the cell, and details of the tools and other equipment in the cell. In addition the temporal status information from the sensors in the cell is stored locally.

Mostly distributed, heterogeneous databases are used for storing the data related to the various sub-systems. This has resulted because of the bottom up evolution of industrial automation [Sepe87]. The choice of the model depends on the nature of the data and the type of queries used to retrieve data. For example, the bill of materials or an indented parts list is implemented using the hierarchical model. Customer billing is done on a relational system.

None of the traditional data models like relational and hierarchal are adequate to represent all the types of data generated at the various stages from design of product to manufacturing and sales. The object oriented approach to modeling helps to naturally depict the entire system. Research in object oriented techniques has led to the evolution of the object oriented DBMS (OODBMS). These OODBMS support an object oriented model. We suggest that the OODBMS will enable natural representation of data in the manufacturing world.

2.2.5 A Typical FMS Scenario

This is a description of a typical scene for a part flow within a FMS. The FCS reads the daily schedule generated by the supervisory controller. It issues requests to the stores for the materials required. The MHSs are directed to prepare the fixtures and tool planning is done.

The FCS retrieves the route sheet from the database for the various types of parts to be manufactured. The FCS decides the routing based on rules and scheduling policies built into it. Real-time scheduling policy may be incorporated into the FCS which may base its decision on the state of the system such as failed machines, raw material availability, and fixtures availability. The cell controller decides on which machine to load the part and retrieves the part programs and tool descriptions required for the particular operation. It routes the parts through the cell, while monitoring their progress through the cell and either reports it to the FCS or stores it in the local database. During this process various sensors and automatic inspection units in the cell also send status information to the cell controller. Once a cell has finished processing a part the FCS guides the part to the next cell through the MHS. In the situation of the next cell being full, the part is temporarily stored in a temporary storage. After all the operations are over, the part is sent to storage. The FCS also takes decisions about the dispatching of AGVs to transport the parts.

2.3 Manufacturing data

We have seen in the previous section that data stored in a manufacturing database is heterogeneous in nature. In this section we aim to highlight the following issues related to this data.

- Magnitude of the data
- Complexity of the data
- Transient nature of the status data

As explained in Chapter 1, data in the manufacturing domain can be classified into four categories,that is, the Structural or "static" data, Status data or Temporal data, Performance data, and Control data.

These different kinds of data can be represented in a homogeneous way using the object representation. The configuration data, product data, equipment data, inventory information, customer information, and purchase details constitute information that is relatively static. Each of them constitutes an object, with a set of attributes to describe them. The status data is basically the temporal values of attributes or *states* of the static object. The performance data is derived from the temporal data. The control data can also be represented as an *object* with the *settings* as an attribute. We describe below the nature of the data associated with some objects such as lots, processing units, material handlers, inventory, tools, and storage units. We observe that more than one category of data is associated with each object.

- Lot information

  Lots are groups of one or more similar workpieces which undergo the same set of operations. This grouping is done to reduce the machine setup times in switching from one type of operation to the other. In an ideal FMS, lot size is 1 unit. A lot moves as a single unit from one machine to the next. It is identified by a unique identifier. With each lot object, the information such as its size, the processing units it has gone through, and the time taken at each processing step are stored. One of the issues is lot renaming. A manufacturing process may split a lot into new smaller lots, or may combine lots together into one lot. Renaming the new lots and keeping track of its predecessor and successor lots may be necessary in this case.

- Processing Units information

  Processing units in a manufacturing cell may range from manual work centers to sophisticated automated machines with controllers and probes.

  Among the structural information stored are configuration details, the operations it can perform, time taken for each of these operations, lot size it can process, setup time, input and output buffer description. The status information includes the state of the processing unit and other temporal measurement parameters. A processing unit could be in the *DOWN* state due to several reasons such as maintenance or shutdown or breakdown or higher level policy decisions. When not in either of *LOADING*, *UNLOADING*, *PROCESSING*, or *DOWN* states, the machine is in the *IDLE* state. A maintenance log and performance parameters like throughput and utility must be maintained to analyze the general "health" of the unit.

- Tool information

  Effective tool management is necessary in flexible manufacturing, since tools are shared by processing units. They have to be scheduled effectively to reduce the waiting time. Tool information comprising of its unique code number, its functions, and its description is stored in the database. Temporal information regarding its present allocation to processing units and its state, whether *IDLE* or *INUSE* is also stored.

- Material Handlers information

  Material handlers cater to the movement of parts and tools among the processing units and the storage units. They may be automated or manually driven and may have a fixed or a flexible routing. Static information regarding material handlers is their identification number, nature of their movement, route details and the state of the material handler which could be in *IDLE*, *LOADING*, *UNLOADING*, *TRANSPORTING*, or *DOWN* state. Performance parameters include utility and rate of transfer.

- Storage Units

  Storage units include the temporary buffers at the processing units, the cell level buffers and the long term storage units located away from the shop floor. The temporary buffers hold workpieces when they are waiting either to be processed by the machines or for the material handlers to transport them after the processing is over. These are the input and output buffers of the processing unit respectively. When their capacities are full processing is blocked.

  The long term storage units are described by the capacity and the environmental conditions in the stores. A storage unit may hold finished products, semi-finished goods, raw materials, currently unused tools, fixtures, and other accessory materials.

- Inventory

  The inventory may include products, semi-finished goods, raw materials, currently unused tools, fixtures, and other accessory materials. The inventory item contains information as to the current levels, maximum capacity for that particular item, storage costs, etc.

  Depending on the purchasing strategy followed, orders are placed for replenishing raw material stock. Some typical policies are listed next [Nath88]. In Just In Time (JIT) policy, reorders are placed only when requirement arises. Reorder point policy places an order for an Economic Order Quantity (EOQ) of raw materials when the inventory levels drop below the critical level. In a system using MRP, the purchase is planned after considering the inventory on hand.

The temporal variations in the plant are captured as the status data of the objects mentioned above. These are the *events* in the plant. Events could be the position of a moving AGV, a change in the temperature of a machine, the starting of processing in a machine, the termination of a process, or breakdown message about a machine. The frequency of occurrence of some of these events is high, and hence, the amount of temporal data generated is huge. The manufacturing data may be of composite type. For example, the data representing position of a vehicle is a set of two coordinates.
In the next subsection we consider the retrieval aspects of the above types of data stored in the manufacturing database.

2.4 Information retrieval

The data that is generated by the manufacturing process is used for controlling the manufacturing operation [Powe87]. Thus the database has to provide for prompt retrieval of the data stored. Temporal data from the plant is used for monitoring the state of the plant at any instant and for maintenance purposes to aid in detecting the cause of a malfunction.

The status data of the various processing units may be retrieved for the following reasons.

- Analysis of production activities: Performance figures are derived from analysis of production activities. These help to understand the system completely, identify the bottlenecks, evaluate the different scheduling policies and analyze the performance of the processes. These also help to generate management reports. The data is thus used to study the *past* information of the system.

- Tracking production operations: The state of manufacturing resources like the machine status, material handler position, inventory levels, lot status, etc. are used to track manufacturing activities. Views may be created to show the operator the status of the entire plant in a given time interval. This data also helps in lot tracking. The temporal data can be used to show how a particular parameter of a machine varied in an interval of time. This is important from the point of view of the operator for analyzing the process. This constitutes the *present* information.

- Planning: The statistics derived from the temporal data can be used to make decisions for the future. For example, a high utilization of a particular machine indicates that the system may soon require additional resources. Thus, decisions for the *future* may be derived from the temporal data.

The temporal data is retrieved from the database and the information derived from it is converted into a format easily understandable by the user. In the situation where the production process is automated, a graphical display of the process aids the operator in controlling and monitoring the plant.

Chapter 3

The MIDAS System

The MIDAS system was developed to handle both data management and control of a flexible manufacturing system. In this chapter we describe the development of MIDAS-1. It involved identifying the entities in the manufacturing domain and modeling the domain, the design of MIDAS-1 system architecture, implementation of the model, and the adaptation of the system to a manufacturing plant example [Bhuv96]. The prototype also incorporates a simple scheduler and several view generation routines. In the next section we describe the various stages in the development of MIDAS-1. Later, we give an outline of the features incorporated in MIDAS-2.

3.1 MIDAS-1

3.1.1 Building the Object Model

The object oriented approach to software development presented in [Rumb+91] is used in the development of MIDAS system. This approach suggests the design of a detailed object model of the system prior to system implementation. Initially, the manufacturing domain was studied and its characteristic features analyzed. Then the requirements of the system were specified. During the analysis, the nature of data and the type of transactions for the manufacturing environment were identified. The various issues to be dealt with while designing the system were listed as the desired features of the system. The data model was designed next. This involved identifying the objects and their attributes and recognizing the inheritance, aggregation and association relations between them. The entities like processing units, storage units, material handlers, lots, and sensors formed the major objects of the manufacturing unit. The details of the complete object model are given in [Bhuv96]. The design of MIDAS-1 system architecture was the next step.

3.1.2 System Design

The architecture of the initial system is shown in Figure 3.1. It comprises of three components, namely, the *Illustra Data Server*, the *MIDAS-1 Server*, and the *MIDAS Server Interface*. A description of each of the three levels of architecture is given below.

- *Illustra Data Server*

  Illustra is a commercial object relational database [IAPI95, IDAG95, IUG95]. It provides support for classes and objects, inheritance, object references and other object oriented features. It uses the relational model for storing the information. Each class in Illustra maps to a table where the instances of the class are stored as records. Most manufacturing systems have their own relational databases in which the manufacturing data is stored. MIDAS can easily support these *legacy* databases since it uses Illustra which is an object relational database. The following features of Illustra which are not common in other commercial DBMSs make it suitable for our application.

  - Support for Objects: Illustra provides an OO interface to the underlying relational model. It provides a mechanism to define complex objects and inheritance of types. Objects in Illustra are of a predefined type and can have composite attributes which may be arrays, sets or a combination of two or more attributes. It does not provide support for member functions, but the user can create a function composed of SQL statements or a C function. Entities in the manufacturing domain that are modeled as complex objects with hierarchical relations can be easily mapped to Illustra objects.

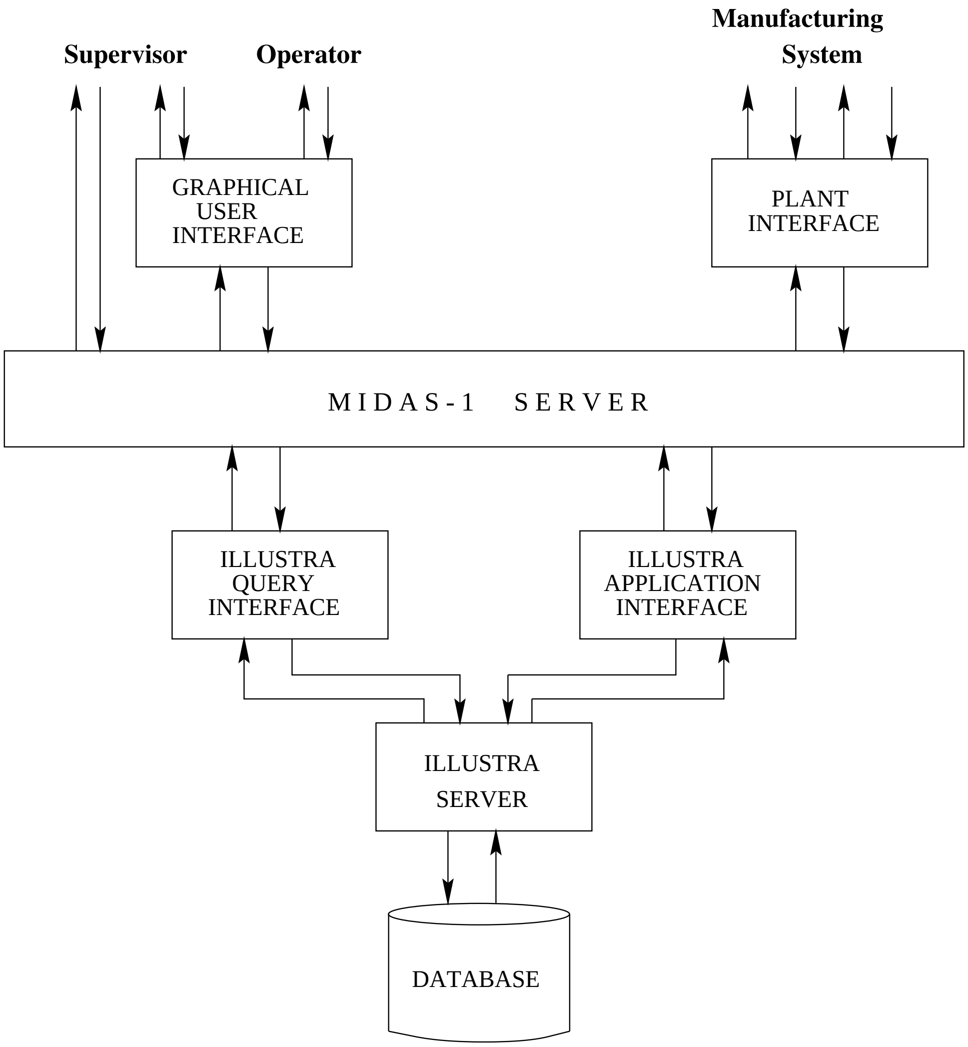


Figure 3.1: MIDAS-1 Architecture

  - Rules: Illustra supports active database features like Rules based on the *Event-Condition-Action* concept. A *Rule* specifies the *Action* to be taken when a *Database event* occurs and the *Condition* is satisfied. Some amount of plant control can be embedded in the database in the form of rules.
  - Isolation Levels for reads: Illustra supports four levels of isolation for reading data. They are "Read Uncommitted", "Read Committed", "Repeatable Read", and "Serializable". Different levels of isolation can be used for different kinds of transactions in the manufacturing domain. For example, status data is modified by plant monitors which we call the "updaters", and it is read by transactions that collect statistics and display the values to the operator which are called the "readers". The updater transactions have to maintain serializability while updating the values but the readers can have lower levels of isolation.
  - Alerters: Illustra has a provision by which it can notify an application about the occurrence of asynchronous events. Alerters can be used to notify the control software of the plant about exceptional events by the database so that corrective action can be taken.
  - Time Travel: This feature enables the user to query data that was current at a point in time or during a period between two points in time. This is made possible by the no-overwrite transaction processing and archiving features of Illustra. The state of the plant at any point in time or during a specified period can be queried, which may be useful for fault analysis.

  The Illustra backend engine forms the lowest layer of MIDAS. It offers two interfaces, the *Interactive Query Interface* and the *Application Programming Interface*. The Interactive Query Interface is a command line interface used for interactive querying on the stored data. The Application Programming Interface contains a set of library functions which may be invoked from application programs.

- *MIDAS-1 Server*

The MIDAS-1 server is essentially the controller of the automated plant. It exploits the active feature of the backend engine to provide control over the plant. It encompasses the decision support routines, all the control routines for plant control and the routines to access the database.

- *MIDAS Server Interfaces*

  The MIDAS Server Interfaces comprise of the *Interactive User Interface* and the *Plant Interface*.The interactive interface is a Graphical User Interface (GUI) developed using X and Motif [Youn90, Moti94]. It provides two views of the plant, the *Supervisory Interface* and the *Customer Interface* which is a restricted view of the plant. The *Plant Interface* is the interface for transferring messages between the plant and the MIDAS server. The Supervisory Interface also has a provision to bypass the GUI. In this case, MIDAS connects the user to the Illustra Query Interface.

3.1.3 Implementation

After completion of the design of Object Model and the System Architecture for MIDAS, the implementation of the MIDAS-1 server and its interfaces was done. The major components implemented in the MIDAS-1 system are described below.

- System Model Implementation

  The entire data model is mapped to the MIDAS database using the Illustra SQL DDL statements. The functions and routines for the plant control are incorporated into the system. Rules and triggers are created and associated with different actions.

- Scheduler

  A scheduler is developed to schedule a set of orders given the due dates and quantity of the product to be delivered for each order. The scheduler gets the information about the capacity of the machines and the process sheet information of the products to be manufactured from the database. Each machine is assigned a criticality number which is proportional to the amount of time the machine is required to be busy. The scheduler fixes a priority to each job based on its machine routing, that is, the sequence of machines it goes through and the criticality numbers of these machines. During processing, the highest priority job among those waiting for a machine is assigned to the machine.

- View Generation

  The temporal data is presented in an easily understandable form to the operator by appropriately combining status data. Three views, namely, plant view, process view, and lot view are provided which can be adapted to a specific plant with minimum modifications.

  - The *Plant View* shows which lots are at each processing unit for the specified time interval.
  - The *Process View* shows the process status of a processing unit while a batch was being produced. It helps the operator to analyze the effect of the process parameters on the quality of the product and to diagnose any defects in the process.
  - The *Lot View* shows the status of every process that the lot passed through. It helps in lot tracking.

3.2 MIDAS-2

The basic MIDAS system is enhanced during the second phase of the project. The Dynamic Model is constructed using the design principles suggested in [Rumb+91]. It depicts the flow of control through the various controllers of the automated units in the manufacturing plant. A scheduler that can produce near-optimal schedules for a reasonably sized manufacturing plant is implemented. It is based on the Improved Lagrangian Relaxation Technique suggested in [CL94]. This, along with the heuristic scheduler discussed earlier, provides more effective Decision Support. The temporal data in the plant has to be acquired and served for display to the operators in real-time. For this purpose, a Real-Time Data Server as proposed in [Shim+93] is incorporated in MIDAS-2. The architecture of MIDAS-2 is shown in Figure 3.2

The improvements made to the previous version of MIDAS architecture are outlined below.

- The *MIDAS-2 Server* incorporates the Lagrangian scheduler. The requirements of a specific manufacturing plant may demand the use of one or more kinds of schedulers to be used at different levels of plant control. Thus, MIDAS now allows the user to choose the appropriate scheduler for scheduling a set of orders.

- Temporal data from the plant that changes very rapidly is received by the *Real-Time Data Server* module instead of directly going to the MIDAS server. The server caches this data to provide real-time access to the GUI tools that display this information to the plant operator. The GUI tools request the real-time server for the periodic transmission of the temporal data. The plant maintenance personnel can also request the server for temporal data within a period of time. The real-time data server is interfaced to the database through the MIDAS server. It makes use of the routines to access the database present in the server to store the temporal data in the MIDAS database.

In the following chapters a detailed description of the above features is given. Chapter 4 describes how the Dynamic Model depicts the major control aspects of the FMS control system. Chapters 5 and 6 give the details of the implementation of the Lagrangian Scheduler and the Real-Time Data Server respectively. Chapters 7 and 8 describe the adaptation of the MIDAS system for a simulated Semiconductor Plant and the details of the GUI provided for the operator respectively.

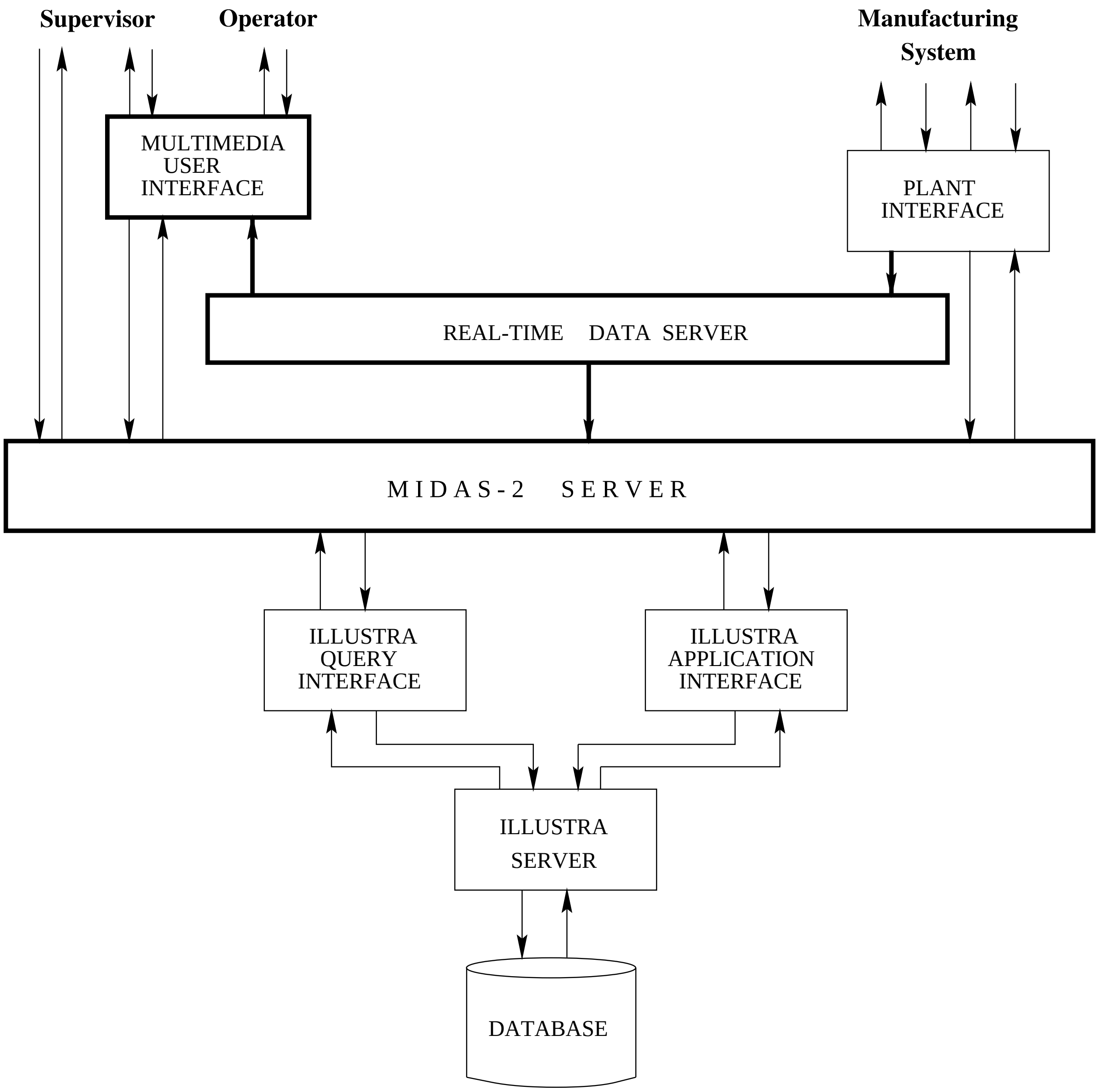


Figure 3.2: Architecture of MIDAS-2

Chapter 4

Dynamic Modeling

As mentioned earlier, the *static* structure of the entities in the manufacturing domain is depicted in the initial MIDAS object model, which was designed as part of MIDAS-1 [Bhuv96]. These objects change their *state* over time during the manufacturing process. The state of an object consists of the attribute values and links held by it. For example, a machine may change its state from “idle” to “setup” when a part is to be processed, the location of the transporter is continually changing while it is moving and the description of a lot changes as it is processed. These state changes are a result of the various operations going on in the plant. The timing and sequencing of the operations is not depicted in the object model. Since MIDAS is intended to participate in the factory control, there is a need to understand the flow of control among the various objects in the manufacturing domain. The Object Modeling Technique (OMT) suggests an augmented model called the *dynamic model* for representing the control aspects of a system. We show the sequencing of various activities in the shop floor by constructing the dynamic model for the manufacturing system. The dynamic model is constructed for a generic automated plant and includes the typical interactions in the shop floor. The control flow between objects in a specific plant can be captured by a subset of the dynamic model. This specific model can be used to easily incorporate control of the intended plant in the MIDAS system.

In this chapter, we first describe the concepts involved in constructing the dynamic model for any system which is a summary of the details given in [Rumb+91]. Later, we describe the steps followed in building the dynamic model for the manufacturing domain.

4.1 The Dynamic Model

The Dynamic model shows the time-dependent behavior of an interactive system. Objects in such a system are concurrently active, sending events to other objects and changing their own state (called state transition) in response to events received. The pattern of events, states, and state transitions for a given class is represented as a *state diagram*. The state diagram for a class represents the behavior of the class in response to events. The *dynamic model* consists of multiple state diagrams, one state diagram for each class with important dynamic behavior.

The first step in constructing the dynamic model is to identify typical scenarios under normal conditions and with exceptions. A *scenario* or an event trace is a sequence of events that occur in a particular execution of the system. For example, in the manufacturing domain all the control messages sent between the various controllers to route a particular lot through the system form a scenario. The next step is to extract events from the scenarios, and to assign each event to its target object. After this is done, the state diagrams for each class are constructed. The basic constructs used in preparing a state diagram for a class are described below:

- *State*: A state is an abstraction of the attribute values and links of an object. Sets of values are grouped together into a state according to properties that affect the gross behavior of the object. A state specifies the response of an object to input events. The object may change its state in response to an event, which is called a *state transition*. Change of state is an instantaneous operation with respect to the granularity of the model (level of modeling).

- *Event*: An event is an occurrence that results in a change to the state of the system. Each object can stimulate the other through sending of events. Events that have common structure and behavior can be classified into event classes. Some event classes may be simply signals that inform that something has occurred, while other event classes convey data values which are called its “attributes”.

- *Condition*: A condition is a boolean function of object values used as a *guard* on the state transitions. When a condition is specified on a state transition, the object will undergo that transition only when the condition is satisfied. When an event is received by an object its response can depend on one or more conditions.

- *Activity*: An activity is an operation that takes time to complete. An object can perform a continuous activity until it receives an event that changes its state, or the activity could terminate after a sequence of operations.

- *Action*: Actions are instantaneous operations whose duration is insignificant compared to the resolution of the state diagram. They are associated with events.

A state diagram is a graph whose nodes are states and directed arcs are transitions labeled by event names, conditions and actions. It gives the sequence of states that an object goes through in response to all the normal and exceptional sequences of events that can occur.

The above basic constructs are not sufficient to represent in a structured format, all the activities of a complex system such as a manufacturing plant. Hence, we describe below, *Nested State Diagrams* that allow generalization of events and states, and constructs to represent *Concurrent* states of an object, which are the advanced dynamic modeling constructs suggested in [Rumb+91].

Nested State Diagrams

All complex systems contain a large amount of redundancy that can be used to simplify state diagrams, provided appropriate mechanisms are available. Nested state diagrams allow events and activities to be expanded into lower level state diagrams. Another form of nesting allowed is generalization of states and events.

- State Generalization: States may have *substates* that inherit the transitions of their superstates. Any transition or action that applies to a state applies to all its substates unless overridden by an equivalent transition on the substate. It allows states that have a common behavior in response to certain events to be grouped into a superstate. The resulting state diagram is structured and thus, is easily understandable.

- Event Generalization: Events can be organized into a generalization hierarchy with inheritance of event attributes. The leaf level of the hierarchy contains the actual events, and the higher level events are abstractions of these events. These abstract events can be used to simplify the state diagrams of the objects with same behavior for a group of events.

Concurrency of States

The dynamic model describes a set of concurrent objects, each with its own state and state diagram. The state of the entire system at any instant is an aggregate of the states of all the objects in the system. Each object can change its state independent of other objects in the system, except in the case of objects that form an aggregation. The states of objects that are parts of an assembly are dependent on the state of the aggregate object. Also, a single object can be in multiple states at the same time.

- Aggregation Concurrency: A state diagram for an assembly is a collection of state diagrams, one for each component. The state of the "aggregate" object corresponds to a collection of states, one for each of the "component" objects. The states of the components may interact, such that one component object cannot be in a particular state unless another is in a specified state. This interaction can be represented using guarded transitions on the first object requiring the latter object to be in a given state.

- Concurrency within an Object: Concurrency within the state of a single object arises when the object can be partitioned into subsets of attributes and links, each of which has its own subdiagram. The state of the object comprises of one state from each subdiagram. The same event can cause transitions in more than one subdiagram.

Some of the advanced dynamic modeling concepts which were useful for modeling the manufacturing domain are listed below.

- Automatic Transitions: Automatic transitions, are transitions from a state without an event specification. They are fired as soon as the activity in the state is completed. These are used to represent a sequence of operations taking place in succession.
- Sending Events: An object can perform the "action" of sending an event to another object.
- Splitting and Merging Transitions: Sometimes an object performs concurrent activities, but both activities need to be completed before it can proceed to the next state. The object may perform each of the concurrent activities on the occurrence of two independent events affecting the same state. Thus, there can be a *forking of control* into concurrent activities and later a *joining of control*.

4.2 Modeling the Manufacturing Domain

The design procedure suggested in [Rumb+91] is used to develop the dynamic model for the manufacturing domain. First, typical scenarios in the manufacturing plant were prepared, and the events were identified and associated with the objects. Then, the state diagrams were developed for each class with significant dynamic behavior.

4.2.1 Preparing Scenarios

A *scenario* is a sequence of events that occur in a particular execution of the system. The major activity in a manufacturing plant is the processing of parts. Hence, the most

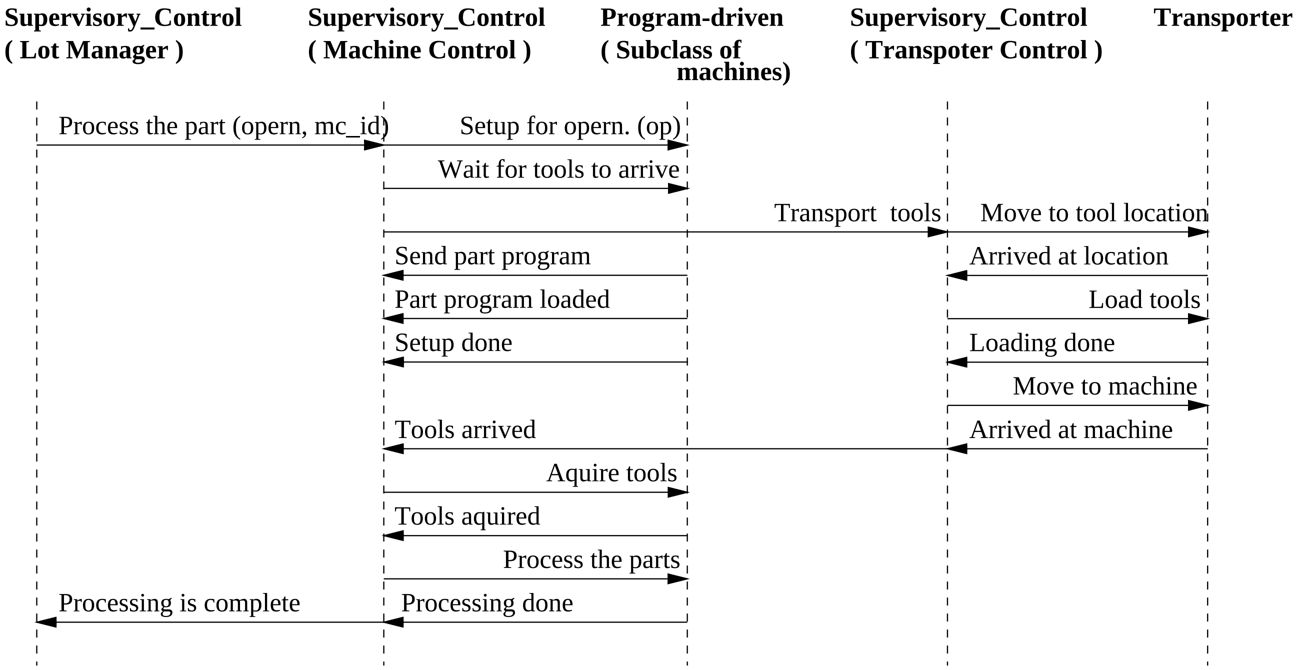


Figure 4.1: Lot Processing by a Programmable Machine

common scenario is the routing of parts through the shop floor. The supervisory controller initiates part processing based on the planned schedule. It issues commands to the automated machines and material handlers to route the parts through the appropriate cells. The controllers of the automated equipment receive these commands and drive the equipment to perform the required job. In addition, they send status information to the higher level controllers. In the MIDAS object model all the controllers above the equipment controllers are abstracted as the "supervisory_control" class. Thus all the events shown in the event trace going from and to the supervisory_control class may in reality be received and sent by one of controllers in the control hierarchy.

The scenarios are constructed using descriptions of the sequence of operations in an FMS with automated machinery and material handlers given in [VN92, Lugg91]. We show in Figure 4.1 a typical scenario of a lot going through a single processing step after it has been dispatched to a program-driven machine.

The supervisory_controller checks for the type of operation to be performed and the part type to be processed, and decides that the configuration of the machine has to be changed. The part program is downloaded from the database to the machine controller.

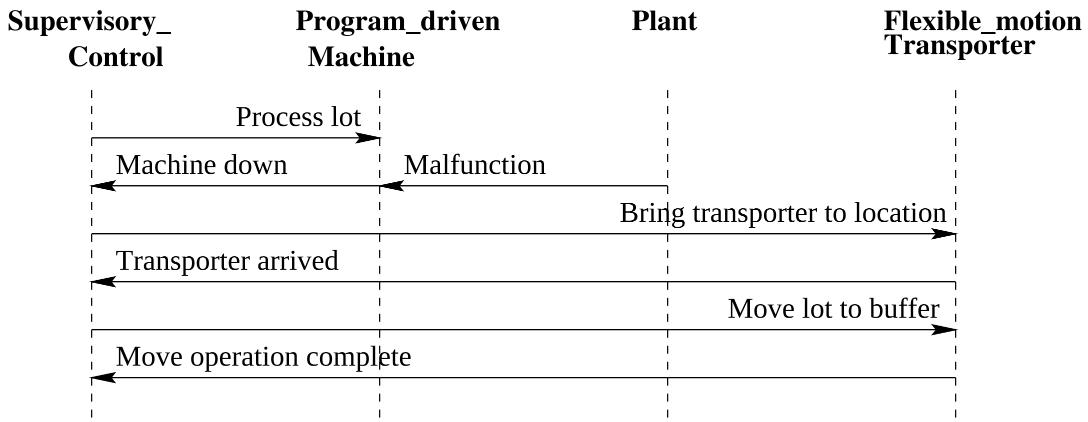


Figure 4.2: Machine Failure during Processing

The supervisory_controller engages a transporter to bring the tools to the required machine where they are loaded onto the tool magazine of the machine. After getting the ready signal from the machine controller, it sends a signal to the machine to start processing. The machine controller now takes over and after completing the processing of the lot, it sends a "processing done" message to the supervisory_controller. The supervisory_controller then sends the lot to the unload station where it is transported to the next processing unit.

The supervisory and cell level controllers also deal with equipment failures and other obstructions to normal operation of the plant. A scenario in which the machine fails due to any of the reasons like tool breakage, part misalignment, machine malfunction, etc., is shown in Figure 4.2. In this case, the supervisory_controller takes a decision to store the lot in a temporary storage unit, and dispatches a transporter for this purpose. The transporter could be a robot that can unload the parts from the machine. This lot could be later scheduled on another machine if rework is possible, or discarded. Expert systems and other decision making policies may be incorporated in the supervisory_controller, in which case it can deal with many other types of failures without the need for human intervention.

There are numerous other event traces in the manufacturing process, out of which we have described a typical sequence of events during normal operation and for a situation with failures.

4.2.2 Identifying Events

Typical events are start operation, halt unit, poll status of sensors, processing done, part arrival, request for part program and part defect. The above events constitute the control signals and signals informing changes in the status of the plant. In addition, equipment failures, obstruction to transporters, etc., are unexpected events. Events can be classified as *control signals*, *status signals*, and *external events*.

- control signals: These are the signals sent by the controllers to request a particular task to be done. Examples are the poll status, start unit, halt unit, request for part program, and dispatch transporter events.
- status signals: Status signals inform the higher level controllers of the state of the unit. Examples are part defect, processing done, transportation complete, tools acquired, and machine down signals.
- external events: Events that are not generated by any of the controllers but are external to the plant control, like tool breakage, transporter failure, part misalignment, and part defective can be considered to be sent from the plant.

An *event flow diagram* summarizes all the events between the classes without regard for sequence. The event flow diagram for the manufacturing domain showing the major events sent across the classes with significant dynamic behavior is given in Figure 4.3.

4.2.3 Constructing State Diagrams

The state diagram for an object shows the events the object sends and receives and how it reacts to the events received. In the rest of this subsection we give a description of the state diagrams built for some of the major classes. The notation used in constructing the state diagrams is suggested by [Rumb+91] and is given in Appendix A.

A computer-numerically-controlled (CNC) machine has a controller that drives the machine to perform a particular operation with the aid of a "part program". It may also have additional logic for automatic probing, broken tool detection, etc. [Lugg91].

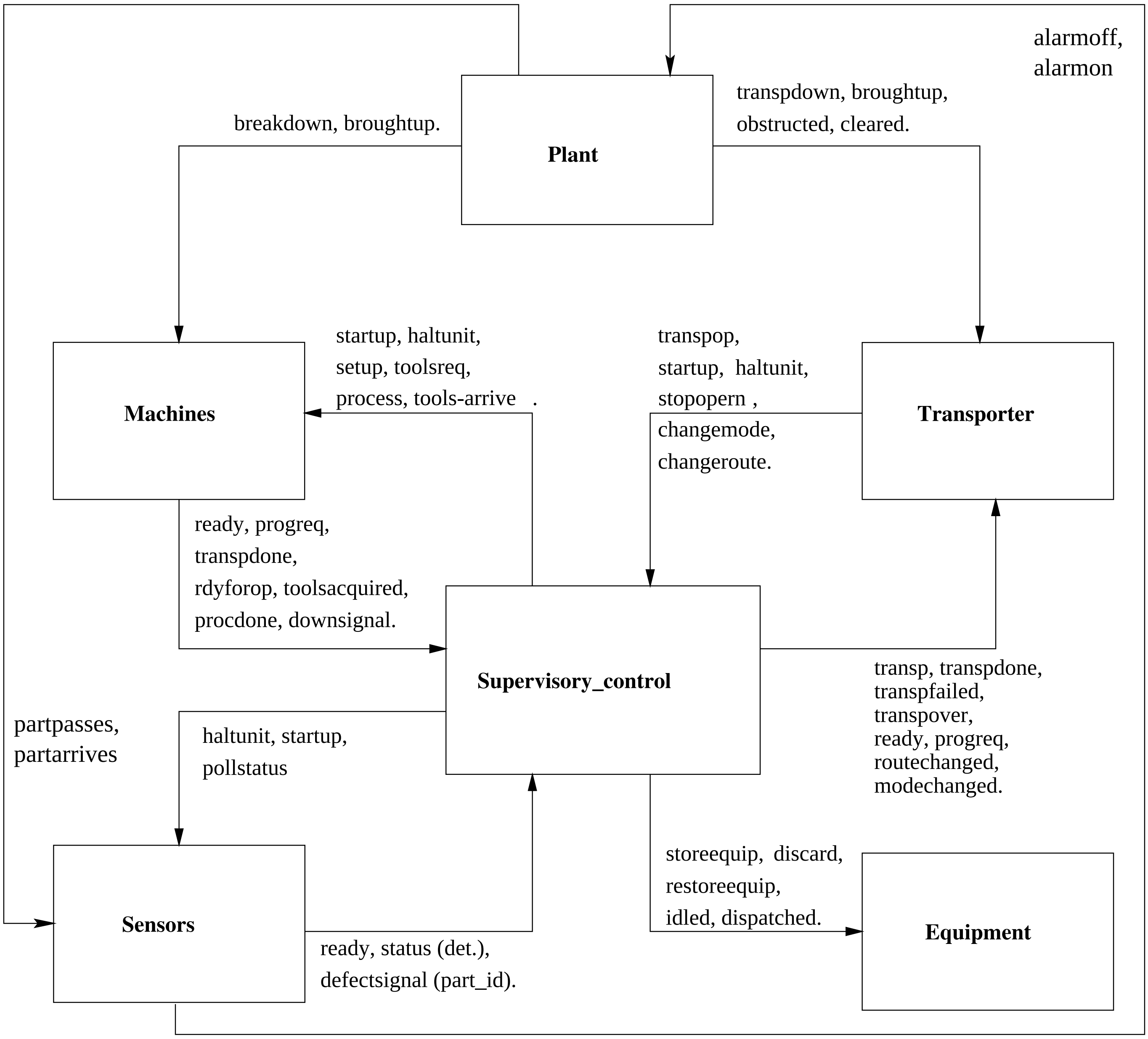


Figure 4.3: Event Flow Diagram for the manufacturing domain

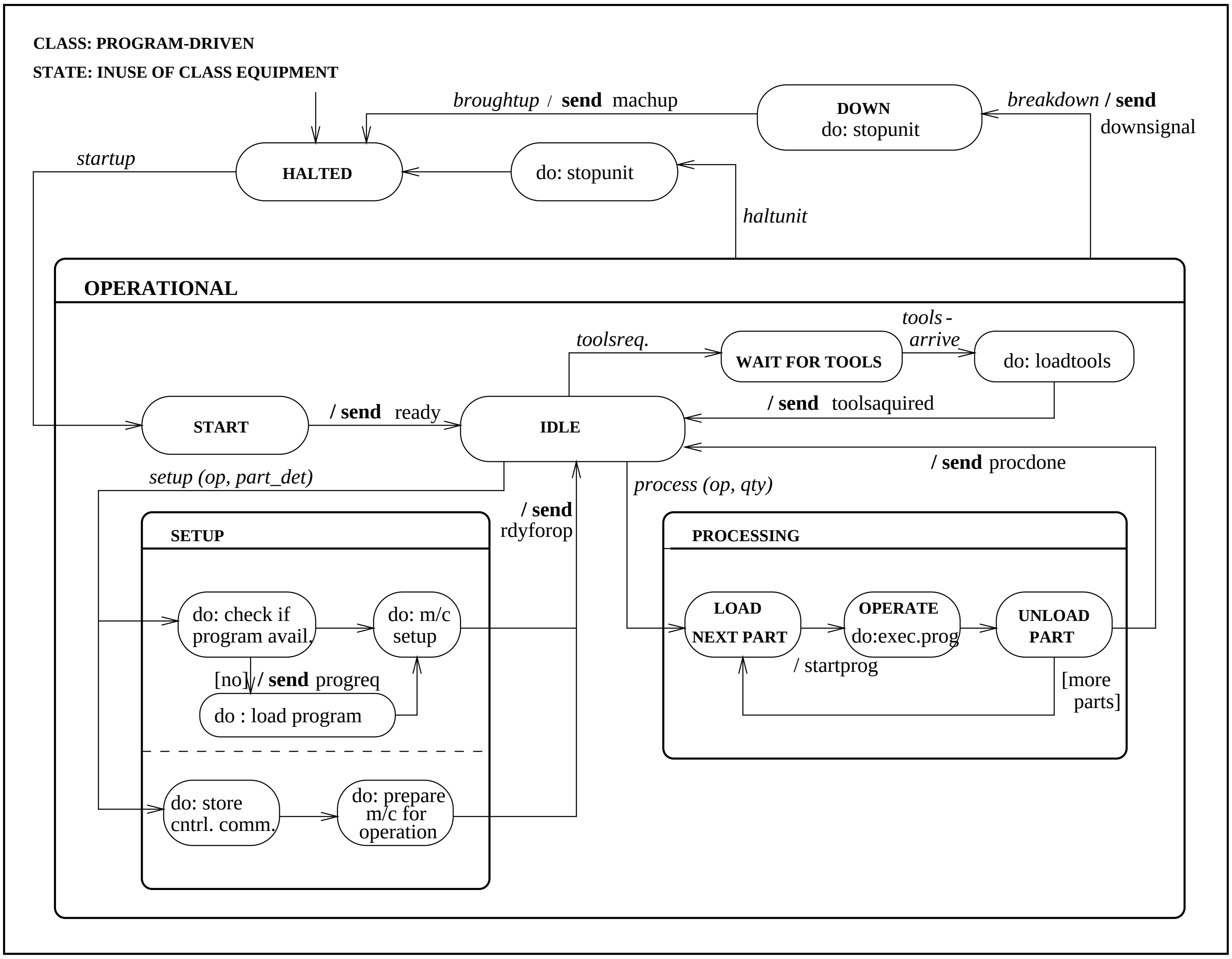


Figure 4.4: Program_driven Class of Machines

Figure 4.4 shows the state diagram for the program driven class of machines that represent the CNC machines. Since, there is an aggregation of controllers and manual controls, the state "setup" of the machine is a concurrent state of both the types of control units. Some of the transitions result in the sending of events to the supervisory_controller like "procdone", "downsignal", etc.. The state "processing" is a nested state consisting of a set of states in which the machine processes each lot_unit using the same program repeatedly. After the processing is done, the state changes automatically to "idle". This is an example of an automatic transition.

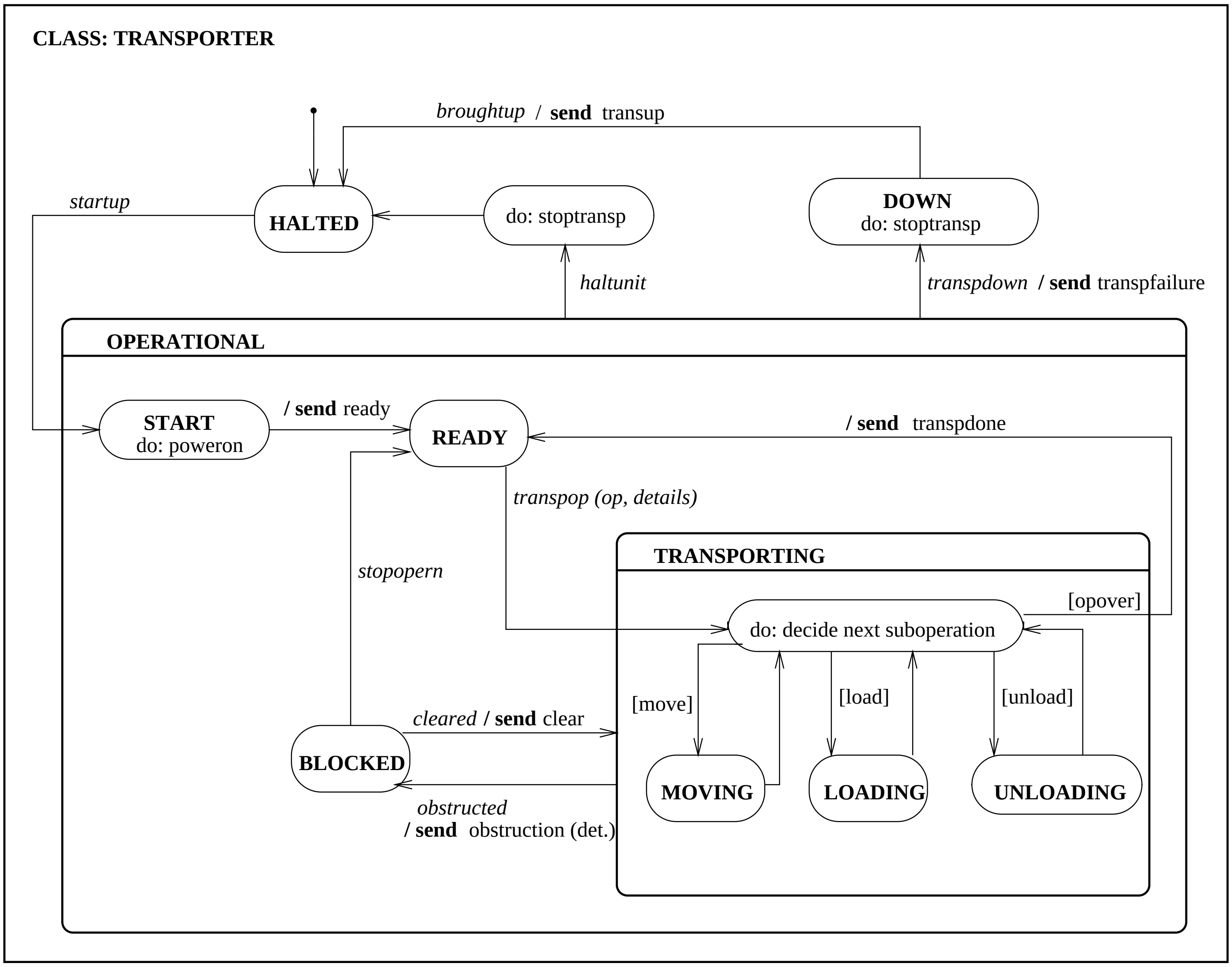


Figure 4.5: Transporter Class

The next major class is that of transporter. The state diagram for the transporter class is given in Figure 4.5. Here we can see that the transporter could be blocked in any one of the three transporting states. The obstruction may clear by itself after some time in which case the transporter resumes, or else it may be given a stop signal by the supervisory controller. In this case, it resumes operation only on the supervisory_ controller's command.

The supervisory_control class encompasses all the controllers higher than the equipment controllers in the control hierarchy of an FMS. Its duties have been logically subdivided into lot management, transporter control, machine control, status monitoring and exception handling. Each of these functions are supported concurrently by the controllers, hence they each have a separate state transition diagram. For example the lot manager is shown in Figure 4.6. Whenever a new lot is created the responsibility of routing the lot is handled by a separate thread of control in the lot manager. The process sheet and the scheduling information for the lot is retrieved by this controller, which based on this information decides the next operation for the lot. Whenever a new lot is created, a separate thread of control is created to route it through the plant. Also, if a lot is made of several lots the new lot has to wait for the component lots to arrive. For every lot, the next operation is decided and a transporter or a machine is dispatched to do the respective job. In the case where there are no more machines available or transporters available the lot manager can temporarily store the lot in a buffer, or leave it in its present location until the resource becomes available. The machine control, transporter control and status monitoring sections of the supervisory_controller work in a similar fashion. The exception handling and machine, transporter failure handling logic of the controller is specific to the plant, hence the details of exception and failure handling are not shown in the model.

On lines similar to those described for the program_driven, supervisory_control and transporter classes, we have developed the complete dynamic model. The dynamic model checked for all the event traces given in the scenarios is given in Appendix B.

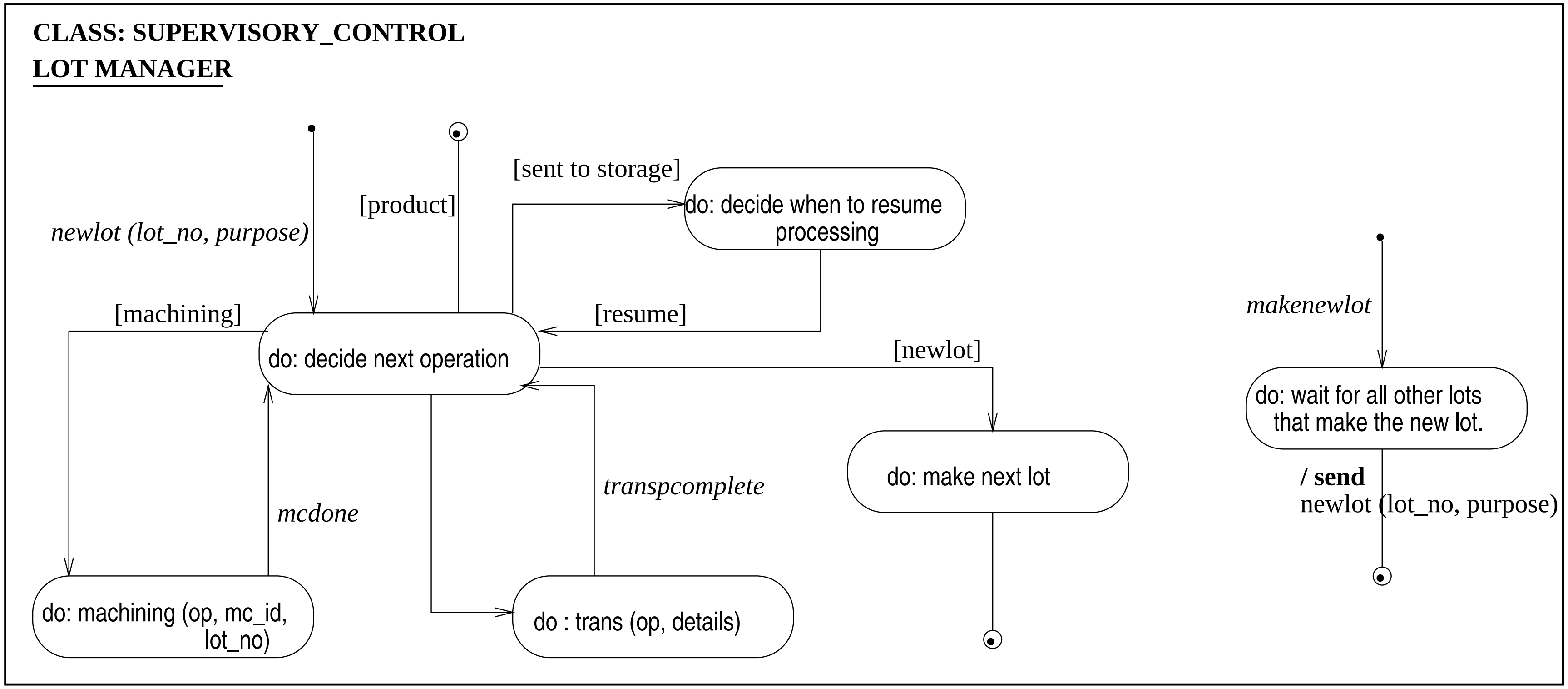


Figure 4.6: Lot Manager

Chapter 5

Decision Support

One of the main goals of an FMS is to achieve maximum "flexibility". Flexibility is expected to prolong the service life of a manufacturing facility and enable it to quickly and economically respond to dynamic changes in product type and mix. Most FMS installations still follow the conventional job shop loading or fixed routing where each part type has to follow only one route through the system. Hence, they fail to utilize the available flexibility [CC91]. Since, typically the machines in an FMS will be capable of performing more than one type of operation, and multiple machines can perform the same operation, each part type can potentially have more than one route through the system, out of which any one route could be followed to process the part. Usually, the problem of scheduling is subdivided into the loading (or routing) problem and the dispatching problem. The loading decision is made to fix the machine sequences for a set of jobs so that they satisfy the process (technological) constraints as given in the process sheet for the product. At each machine, the choice of the jobs to be loaded from those waiting to be produced depends on the dispatching decision. This technique does not capture the interaction among the jobs that are part of the manufacture of a single product. Thus, the flexibility, complexity and the need for system integration makes the problem of scheduling for an FMS more difficult. Therefore, effective scheduling schemes are required to maximize system throughput.

The problem of finding an optimal schedule for the manufacture of a set of products with multiple possible routings is known to be NP-hard [LRB77]. Mathematical techniques like branch and bound and dynamic programming techniques may require exponential time in the worst case. Polynomial time approximation schemes for solving scheduling problems which produce sub-optimal schedules within polynomial time have been examined. These techniques are rarely used in practical situations where there are hundreds of machines and products.

There are a number of ideas developed to deal with the complexity of scheduling for FMS [CC91, CB90]. Several *heuristic algorithms* to generate "good" but not always optimal solutions have been suggested. These heuristics are used in combination with Material Requirements Planning. They do not produce schedules of consistent performance, since parts are sequenced according to rules that do not account for the relations among parts of a product established by the MRP system. The advantages of heuristic schedulers are that they are more efficient to compute and can easily react to dynamic changes.

MIDAS-1 includes a heuristic scheduler based on the algorithm suggested in [DJ90]. The scheduling logic involves finding out a priority number for each job based on the job slack and machine criticality factors. Job slack is the time available for a job before its due date besides the processing time required. A machine that will be utilized for more amount of time is given a higher criticality factor. These job priorities are used in deciding which job among a set of waiting jobs is selected for processing by a machine.

In MIDAS-2 we augment the decision support system with a scheduler based on a mathematical technique called "Lagrangian Relaxation" that can solve large scheduling problems within reasonable amount of time, and can adapt to dynamic changes. This technique makes use of the special structure in the scheduling problem to decompose it and reduce its complexity. The solution obtained is *near-optimal*, that is the cost of the schedule is within a small percentage of the optimal (or minimum) cost. In addition, the performance of the resulting schedule can be measured with respect to the lower bound on the cost of the schedule obtained after solving the dual problem. In this chapter, we first introduce and present the use of this technique for the FMS scheduling problem as developed by [CL94]. Later, we discuss the details of the integration of this scheduler into the MIDAS server.

5.1 The Scheduling Problem

In this section, we define the problem of integrally scheduling products with Bills of Materials in order to reduce the work-in-process inventory and to be able to accurately predict and control the job completion times. The bill of materials for a product is a hierarchical arrangement of parts that gives the details of the raw material requirements of the product and specifies the order in which the parts are to be processed or assembled. The manufacture of each part involves a sequence of operations. The order in which these operations can be performed is given by the "operation precedence constraints", which define a partial order on the set of operations. Similarly, constraints on the order of processing or assembly of parts to make a product as dictated by the bill of materials for the product make the "part precedence constraints". The other limitation is the availability of machines to process the jobs. The scheduling problem is to find a schedule which minimizes the average lateness or "tardiness" of the jobs in meeting their due dates and the work-in-process inventory subject to the above mentioned constraints. Work-in-process inventory is the inventory of partly finished products that are in the shop floor during various stages of production.

5.2 The Lagrangian Relaxation Technique

Lagrangian Relaxation is a mathematical programming technique for solving constrained optimization problems. The idea behind the technique is based on the observation that many hard problems (non-polynomial time complexity) like the scheduling problem, are complicated by a small set of side constraints without which the problem is solvable in polynomial time. We introduce the technique as given in [Fish81] by applying it to a combinatorial problem formulated as an integer program similar to the scheduling problem. Consider the integer programming problem given as:

$$Z = \min\ cx,\ \text{s.t Ax} = \text{b},\quad \text{Dx} \leq \text{e},\quad \text{x} \geq 0\ \text{ and integral} \tag{5.1}$$

where x is $n \times 1$, b is $m \times 1$, e is $k \times 1$ and all other matrices have appropriately matching dimensions. The Lagrangian problem after relaxing the inequality constraints is given as:

$$Z_D(u) = \min\ cx + u(Dx - e),\quad Ax = b,\quad x \geq 0 \text{ and integral} \tag{5.2}$$

where $u = (u_1, \cdots, u_m)$ is a vector of Lagrangian multipliers.

For convenience we assume that Equation 5.1 is feasible and that the set $X = \{x | Ax = b,\ x \geq 0 \text{ and integral}\}$ of feasible solutions is finite. By assuming an optimal solution x^* to Equation 5.1 and observing that

$$Z_D(u) \leq cx^* + u(Dx^* - e) \leq Z$$

and requiring $u \geq 0$, it can be seen that $Z_D(u) \leq Z$. The fact that $Z_D(u) \leq Z$ allows Equation 5.2 to be used to find lower bounds. We use the lower bound to quantify the performance of the final feasible schedule.

In the following sections we see the Lagrangian Relaxation technique applied in relaxing the constraints for the scheduling problem. The scheduling problem is complicated by the set of constraints that specify the interrelations between the parts that make a product and the order of operations required to produce a part. The relaxed problem is decomposed and can be solved in linear time. The integer programming formulation for the scheduling problem is given in the next section.

5.3 Problem Formulation

The objectives of the scheduling which are to improve the on-time part completion and to reduce the work-in-process inventory, are modeled by the weighted part tardiness and earliness penalties. Each product to be scheduled is assigned a “weight” that gives the importance of the job with respect to all other jobs. The earliness penalties are used to penalize the early starting of a job with later due date, which could result in accumulation of work-in-process inventory. The schedule generated should obey the part precedence and the operation precedence constraints. Also, the number of machines available for processing at any instant is limited and the jobs require certain amount of processing time continuously on the machine on which they are processed.

For representing the scheduling problem as an integer programming problem, the time available for scheduling is divided into discrete time slots. The maximum available time for scheduling is given by the number of discrete time units K. The machine availabilities are represented as the number of machines available at every time unit starting from 1 to K. Assume that there are P products to be processed and product p, $1 \leq p \leq P$, contains N_p parts, where part i of product p is denoted by (p, i). Each part requires a sequence of N_{pi} operations where operation j of part (p, i), denoted by (p, i, j), can be processed by a machine from the set of machine types H_{pij}. The machine that is selected for the operation (p, i, j) is denoted by $m_{pij} \in H_{pij}$. The relative priorities of the products is reflected by the weight assigned to each product p, denoted by W_p. The mathematical formulation of the integer programming problem for scheduling products is given in the next two subsections.

5.3.1 The Objective function

Additional penalty functions are introduced in the objective function to reduce the solution oscillation associated with the search techniques used to solve the problem [CL94]. The penalty functions that make up the final objective functions are given below.

a) *Product tardiness penalty:* The tardiness of a product p is defined as the amount of time by which the product completion time C_p exceeds the due date of the product D_p, that is, $\max[0, C_p - D_p]$. The Product tardiness penalty is the weight W_p times the square of tardiness T_p. Quadratic tardiness penalties reflect that a job becomes more critical with each time unit after crossing the due date. Thus if uniform weights are used, the penalty of two jobs being late by one time unit is lesser than that of a single job being late by two time units.

b) *Product earliness penalty:* The low work-in-process requirement is captured by penalizing the early start of a product. The early start date s_p can be roughly estimated based on the its due date and the time required for manufacturing the product.

$$s_p \equiv D_p - \gamma \sum_{ij} t_{pij}, \ 1 \leq p \leq P; \ \gamma > 1,$$

where t_{pij} is the processing time of operation (p, i, j) and γ is a coefficient greater than 1. The product earliness penalty E_p is defined as product of the weight β_p and $\max[0, \ s_p - B_p]$, where B_p is the beginning time of the product.

The product tardiness and earliness penalties reflect the actual cost of a schedule in terms of missed due dates and accumulating work-in-process. The objective function that reflects these costs is

$$J \equiv \sum_{p=1}^{N} (W_p T_p^2 + \beta_p E_p^2)$$

.

In order to reduce the solution oscillations resulting due to the use of Lagrangian multipliers to solve the dual problem, the following penalties are introduced into the final objective function. The details of this phenomena of oscillations is given in [CL94].

c) *Tardiness and Earliness penalties for parts and operations:* The tardiness for part (p, i), T_{pi}, is defined as the amount by which the completion time c_{pi} exceeds the part due date d_{pi}, that is, $\max[0, \ c_{pi} - d_{pi}]$, and the part earliness is defined as $\max[0, \ s_{pi} - b_{pi}]$ where $s_{pi} \equiv d_{pi} - \gamma \sum_j t_{pij}, \ \gamma > 1$. The operation penalties are defined in a similar manner. These penalties are expressed as

$$\sum_{pi} (w_{pi} T_{pi}^2 + \beta_{pi} E_{pi}^2) + \sum_{pij} (w_{pij} T_{pij}^2 + \beta_{pij} E_{pij}^2), \ \ 1 \leq p \leq P; \ 1 \leq i \leq N_p; \ 1 \leq j \leq N_{pi}$$

The part and operation due dates are selected by performing backward scheduling from the product due date D_p to reflect the structure of the bill of materials.

The final objective function to be minimized is the sum of penalty functions stated in a), b) and c) and is given as:

$$J_{AUX} = \sum_{pij} (\overline{W}_{pij} T_{pij}^2 + \overline{\beta}_{pij} E_{pij}^2), \quad 1 \leq p \leq P;\ 1 \leq i \leq N_p; i \leq j \leq N_{pi}, \text{ with}$$

$$\overline{W}_{pij} \equiv w_{pij} + (w_{pi} + W_p \triangle_{pN_p}) \triangle_{piN_{pi}}, \text{ and } \overline{\beta}_{\mathrm{pij}} \equiv \beta_{\mathrm{pij}} + (\beta_{\mathrm{pi}} + \beta_{\mathrm{p}} \triangle_{\mathrm{p1}}) \triangle_{\mathrm{pi1}} \tag{5.3}$$

In the above equation the fact that the beginning time of a product is the same as the first part's beginning time and that its completion time corresponds to that of its last part has been used. As such, $\triangle_{p1}$ is an integer equal to one if part (p, i) is the first in product p and zero otherwise, and $\triangle_{pN_p}$ is similarly defined for the last part in product p. $\triangle_{pi1}$ and $\triangle_{piN_{pi}}$ are similarly defined for the first and last operations of part (p, i) respectively. The tardiness weights are chosen such that $w_{pij} \ll w_{pi} \ll W_p$, since minimizing product tardiness is the foremost criterion. The earliness coefficient β_p is inversely proportional to W_p since a product with high tardiness priority should be allowed to start early. The earliness coefficients satisfy $\beta_{pij} \ll \beta_{pi} \ll \beta_p$, similar to the relation among the tardiness weights.

5.3.2 Constraints

The minimization of Equation 5.3 is subject to the part precedence constraints, operation precedence constraints, machine capacity constraints and the processing time requirements. We formalize these constraints below.

a) *Part precedence constraints:* Parts in the bill of materials for a product may be assembled with other parts after a specific operation of the latter part. The part precedence constraints may require the beginning time of operation r of part q to be greater than or equal to the completion time of part (p, i) plus any required timeout denoted by S_{piq} between part (p, i) and operation r of part q, that is,

$$c_{pi} + S_{piq} + 1 \leq b_{pqr}, \quad 1 \leq p \leq P;\ 1 \leq i \leq N_p - 1;\ (p, q, r) \in I_{pi}$$

I_{pi} in the above equation denotes the set of operations of other parts (not part i) immediately following part (p, i) in the processing sequence of the bill of materials.

b) *Operation precedence constraints:* Let I_{pij} denote the set of operations in part i of product p immediately following operation (p, i, j) in the operation processing sequence. The operation precedence constraints require the beginning times of the set of operations in I_{pij} to be greater than or equal to the completion time of operation (p, i, j) plus any required timeout S_{pijl} between operations (p, i, j) and $(p, i, l), l \in I_{pij}$, that is,

$$c_{pij} + S_{pijl} + 1 \leq b_{pil},\ 1 \leq p \leq P;\ 1 \leq i \leq N_p;\ 1 \leq j \leq N_{pi} - 1;\ l \in I_{pij}$$

c) *Machine capacity constraints:* The machine capacity constraints require that the total number of operations active on machine type h at time k to be less than or equal to the number of type h machines available at time k, that is,

$$\sum_{ij} \delta_{pijkh} \leq M_{kh},\ 1 \leq p \leq P,\ 1 \leq i \leq N_p;\ 1 \leq j \leq N_{pi};\ 1 \leq k \leq K;\ 1 \leq h \leq H,$$

where δ_{ijkh} is an integer variable equal to one if operation (p, i, j) is active on machine h at time k and zero otherwise. M_{kh} is the number of type h machines available at time k.

d) *Processing time requirements:* The processing time requirement for operation (p, i, j) states that the elapsed difference between the beginning time b_{pij} and the completion time c_{pij} should be $t_{pi(m_{pij})}$, the required processing time for (p, i, j) on machine type m_{pij} selected for the operation, that is,

$$c_{pij} = b_{pij} + t_{pij(m_{pij})} - 1,\ 1 \leq p \leq P;\ 1 \leq i \leq N_p;\ 1 \leq j \leq N_{pi};$$

Among the above variables, the due dates, processing times, timeouts, number of machines available per type, and the number of machines as a function of time are given. The beginning times b_{pij}, and the machine type selected for each operation m_{pij}, are the decision variables.

5.4 Solution Methodology

The scheduling problem is decomposed after applying the Lagrangian Relaxation to the final problem formulation. The resulting subproblems are solved by enumerating all possible beginning times and all possible machines. The dual problem is solved using the *modified subgradient method* presented in [Came+75]. The resulting solution may not be feasible, in which some of the constraints may be violated. These may be part precedence, operation precedence, or the machine capacity constraints. A heuristic approach that is an extension of the *list scheduling* algorithm presented in [Hoit+90] is used to get the final schedule.

The operation precedence, part precedence and the capacity constraints are relaxed by using the Lagrangian multipliers η_{pijl}, η_{piq}, and π_{kh} respectively. The relaxed problem is

$$R : L \equiv \min_{b_{pij}, m_{pij} \in H_{pij}} [\sum_{pij} (\overline{W}_{pij} T_{pij}^2 + \overline{\beta}_{pij} E_{pij}^2) + \sum_{kh} \pi_{kh} (\sum_{pij} \delta_{pijkh} - M_{kh})$$

$$+ \sum_{pi,(p,q,r) \in I_{pi}} \eta_{piq} (c_{pi} + S_{piq} + 1 - b_{pqr}) + \sum_{pij, l \in I_{pij}} \eta_{pijl} (b_{pij} + t_{pij(m_{pij})} + S_{pijl} - b_{pil})],$$

subject to the processing time requirements. This results in the minimization subproblem for each operation (p, i, j) with π and η given:

$$R_{pij} : L_{pij} \equiv \min_{b_{pij}, m_{pij} \in H_{pij}} [\overline{W}_{pij} T_{pij}^2 + \overline{\beta}_{pij} E_{pij}^2 + \sum_{(p,q,r) \in I_{pi}} \eta_{piq} (b_{pij} + t_{pij(m_{pij})}) \triangle_{piN_{pi}}$$

$$- \sum_{q:(p,i,j) \in I_{pq}} \eta_{pqi} b_{pij} + \sum_{l \in I_{pij}} \eta_{pijl} (b_{pij} + t_{pij(m_{pij})}) - \sum_{l:j \in I_{pil}} \eta_{pilj} b_{pij} + \sum_{k=b_{pij}}^{c_{pij}} \pi_{k(m_{pij})}]. \qquad (5.4)$$

In the above subproblem, the fact that $c_{pi} = b_{pij} + t_{pij(m_{pij})} - 1$ for $j = N_{pi}$ and $\delta_{pijkh} = 0$ for all $h \neq m_{pij}$ have been used. The dual problem is formed by maximizing R with respect to the multipliers:

$$D : \max_{\pi,\eta\geq 0} L(\pi,\eta), \text{ with L}(\pi,\eta) \equiv$$

$$[-\sum_{kh} \pi_{kh} M_{kh} + \sum_{pi,(p,q,r)\in I_{pi}} \eta_{piq} S_{piq} + \sum_{pij,l\in I_{pij}} \eta_{pijl} S_{pijl} + \sum_{pij} L_{pij}]. \tag{5.5}$$

5.4.1 Scheduling Individual Operations

Solving the subproblem R_{pij} in Equation 5.4 entails enumerating all possible beginning times for each possible machine type $m_{pij} \in H_{pij}$, and computing the values of L_{pij} calculated for each possibility. The complexity of the subproblem is therefore of the order $K * |H_{pij}|$. If the earliest start time for a product is given, then the enumeration starts from that time instead of from time unit one.

5.4.2 Solving the Dual Problem

An iterative technique that updates the Lagrangian multipliers so that the solution at each stage approaches the optimal value at a reasonable pace is usually employed to solve the dual problem. The*modified subgradient method* is one such technique that has been proven to have very good convergence properties [KA90]. The method consists of computing the current search direction as a linear combination of the current subgradient and the direction used at the previous step. When an edge is encountered this search direction forms a smaller angle with the direction towards the maximum than does the direction of the subgradient, thus enhancing the speed of convergence. We show the use of this technique to solve the above problem, as given in [CL94].

The Lagrangian multiplier π is updated by

$$\pi^{n+1} = \pi^n + \alpha^n s^n,$$

where α^n is the current step size and s^n is the current search direction. The step size α^n

is given by

$$\alpha^n = \beta \frac{L^U - L^n}{(s^n)^T(s^n)}, \qquad 0 < \beta \leq 1,$$

where L^U is an upper estimate of the optimal solution of Equation 5.5 and L^n is the value of L at the nth iteration. The current search direction s^n is given by

$$s^0 = g(\pi^0), \qquad s^n = g(\pi^n) + \gamma_n s^{n-1}, \; n = 1, 2, \cdots, \tag{5.6}$$

where $g(\pi^n)$ is the current subgradient of L. γ^n is given by:

$$\gamma^n = \max[0, -\epsilon^n \frac{(s^{n-1})^T(g(\pi^n))}{(s^{n-1})^T(s^{n-1})}], \qquad 0 \leq \epsilon \leq 2, \tag{5.7}$$

and s^{n-1} is the direction applied at the previous iteration. The subgradient component of the hth resource at the kth time is equal to

$$\sum_{pij} \delta_{pijkh} - M_{kh}.$$

The equations 5.6 and 5.7 describe the subgradient method only when $\gamma^n > 0$. A similar method is used to update the multipliers η_{piq}, η_{pijl}. The subgradient components for the above two multipliers are:

$$(c_{pi} + S_{piq} + 1 - b_{pqr}) \text{ and } (\mathrm{b_{pij} + t_{pij(m_{pij})} + S_{pijl} - b_{pil}})$$

respectively. This method converges at the rate of geometric progression. The solution to the dual problem at the end of the subgradient algorithm is usually associated with an infeasible schedule. We construct a feasible schedule using a heuristic based on the "list scheduling" concept. We next develop an algorithm which is an extension of the heuristic suggested in [Hoit+90] for finding the feasible schedule.

5.4.3 Greedy Algorithm

The schedule resulting from the subgradient algorithm may be infeasible, that is, any of the capacity, part precedence or operation precedence constraints may be violated. The processing time requirements are always satisfied since these constraints apply to the dual problem as well.

A list is prepared by arranging the jobs in the ascending order of beginning times as obtained from the subgradient method. All the jobs that have the same beginning time are ordered in the decreasing order of "incremental cost". Incremental cost for a job is the amount by which its tardiness penalty increases if it were to be scheduled in the next time slot. Initially, the time of scheduling denoted by *now* is set to the first time slot. At every iteration, the steps given below are followed until all the jobs are scheduled. We assume that the time horizon is large enough to schedule all the jobs.

(1) For the next job, may be (p, i, j) in the list, check if

a) the machine m_{pij} is free for all the time slots from *now* to $(now + t_{pij(m_{pij})})$.

b) the beginning times of all the jobs $(p, q, r) \in I_{piq}$ are greater than or equal to $now + t_{pij(m_{pij})}$. In other words, the part precedence constraints will not be violated for this job if it is scheduled at time *now*.

c) the beginning times of all jobs $(p, i, l),\ l \in I_{pijl}$ are greater than or equal to $now + t_{pij(m_{pij})}$. It means that operation precedence constraints will not be violated if the job is scheduled at time *now*.

d) the earliest start time for the job is given to be less than or equal to *now*.

If any of the above conditions is false, go to step (2), else go to step (4).

(2) If the beginning time of the job is less than or equal to *now* then change its beginning time to the latest time such that none of the conditions b), c) and d) are false, and reorder the list. If not, go to step (3).

(3) If the end of list is reached, go to step (5), else go to step (1).

(4) Schedule the job at time *now* and remove it from the list. If the list is empty, then done, else go to step (1).

(5) Increment *now* by one unit and move to the beginning of the list. Go to step (1).

5.5 Implementation Details

The Lagrangian scheduler described above was implemented and integrated into the MIDAS-2 server. The scheduler requires information about the products, the machine availabilities, parts and operations required for each product and their precedence relations, processing time requirements for each operation, and the priorities for each product. Some of this information, like the products to be manufactured, their due dates and priorities, can be input by the user.

The user is provided a graphical interface from which orders can be entered and one of the two schedulers, that is, either Heuristic or Lagrangian can be chosen to prepare the schedule. The order entry form allows the user to select the product, and give the quantity required, due date, and the priority for the order. Each of these orders is entered into the database by the interface provided. Alternately, these orders could be directly entered into the database bypassing the graphical interface. After a set of orders have been entered, the scheduler could be chosen to generate a schedule for the products ordered. The resulting schedule is stored in the database, so that the shop floor controller can make use of it to dispatch jobs to processing units during production. The implementation and integration of the scheduler involved writing 2500 lines of C++ code.

Some of the inputs to the scheduler are derived from the database, the details of which are given below.

- Each order is divided into a set of suborders for the same product. The quantity of product in each suborder is less than or equal to the optimal batch size (or lot size) for the product, and can thus be treated as a single unit.

- The types of machines and the number of machines capable of each type of operation are obtained from the "operations" class and the "process_sheet".

- The parts that make up the products and their precedence constraints are obtained from the bill of materials information stored as the aggregation relation among workpieces.
- The sequence of operations that are involved in manufacturing a part are given in "sub_operations" relation of the operation class.
- The processing times for each operation on each of the machine types is obtained from information in the "process_sheet" class.

Since time is represented in discrete units starting from the day of scheduling, the operation times and due dates of the product are accordingly converted into integers using the smallest time unit out of all the operation times. The time horizon is chosen such that all the jobs can be scheduled within that time and complexity of the algorithm is not unreasonably large. The heuristic used for this purpose is to divide the total time required for processing all the jobs on each type of machine by the the number of machines of that type available, and find the maximum of this value for all machines types to get the value of K. The weights used for deriving the tardiness and earliness penalties for the products, parts and operations are calculated based on the priority of the order, according to the suggestions in the algorithm [CL94]. The product tardiness weight W_p is the same as the priority of the order. The tardiness weights for parts and operations are calculated using the approximation $w_{pi} \equiv \frac{1}{10}W_p$ and $w_{pij} \equiv \frac{1}{100}W_p$ respectively. Similarly, the earliness penalty weights are chosen as $100\beta_p \equiv \frac{1}{W_p}$, $100\beta_{pij} \equiv 10\beta_{pi} \equiv \beta_p$. These settings reflect that the problem of meeting due dates is given more importance than reducing the WIP inventory while finding a solution to the scheduling problem.

After the scheduler is run, the beginning times B_{pij} and machine type m_{pij} for each operation are obtained. They are used to produce an exact schedule that specifies the particular machine to be used and the beginning and end times for each operation.

Chapter 6

Real-time Data Server

In a manufacturing plant, two kinds of data are collected and displayed to operators. One is temporal data that represents the process values that change continuously and which need to be sampled frequently. Other is event data that changes only when events like lot movement, machine setup and transporter dispatch occur. In MIDAS-2, we provide real-time acquisition and retrieval of temporal data to facilitate effective plant monitoring. Temporal data is sampled at regular intervals so that it can be used in defect, fault and performance analysis. In order for this data to be useful, the sampling interval has to be sufficiently small, and the order of the changes in data is significant for defect analysis. The data is retrieved periodically for immediate display of plant status to the operators. The capability to sample plant temporal data and serve it to display tools in real-time is provided by a Real-time Data Server. The task of acquiring data from the plant and registering it with a time stamp is a hard real-time task, that is, in no case should the interval between any two consecutive samples of data from the plant be more than the specified sampling rate. The time stamp is attached to the data at the time of sampling by the real-time server. Data retrieval for display to operators is a firm real-time task, that is, it is desirable if the data can be served at the required rate, but if a particular data is not served within the interval it is discarded. We give the design of a real-time server presented in [Shim+93] that satisfies the above requirements, in the next few sections. The design is used in developing a real-time data server interfaced to MIDAS database through the MIDAS-2 server.

6.1 Design of the Real-time Server

Temporal data from the plant are represented as a set of variables that are the process values from the plant along with a time stamp corresponding to the time at which the data are sampled. A *series* is a set of temporal data whose time stamps increase in a monotonic rate. Temporal data sampled in a period

$$[b + n.c,\ \ b + (n + 1).c]$$

can be regarded as sampled at time $b + n.c$, where c is the rate of the series.

6.1.1 Tasks in the real-time data server

The tasks required to be performed by the real-time server related to acquiring and registering temporal data from the plant and transmitting this data to the clients that request for the transmission of the temporal data are detailed below.

- Data Acquisition: Process values sampled from the plant are periodically written on the memory within the real-time data server. The real-time server forms temporal data by attaching a time stamp to the process values and registering the data within an acquisition cycle.

- Periodic Read: The software tools that display the plant status are clients to the real-time server that request the periodic transmission of latest temporal datum. On receiving a request from the client, the server starts the periodic transmission of temporal data. The cycle length and the particular data required is specified in the read request. The reading activated by this kind of request will be referred to as "Periodic Reading". Periodic reading is a firm real-time task in which the deadline is the end of a cycle.

- Aperiodic Read: The clients may require to retrieve past temporal data in a specific period. The server composes series of registered temporal data within the required period according to each request. The reads required to process the aperiodic requests are called "request driven reads". Request driven reads do not have deadlines and run as background jobs.

- Receiving requests and Transmitting data: All the messages between the clients and the real-time data server are transmitted using the UDP/IP protocol. Typically, the server may be transmitting data periodically to more than one client, hence the datagram service provided by UDP is most suitable. Connection-oriented protocols like TCP/IP if used will require a connection to be opened and closed for every cycle of the periodic transmission. This is a high overhead since cycle times of a few hundred milliseconds are typically used.

  The clients send messages to start or stop a periodic reading task or to request for a series of data within a specified period. The server acknowledges the periodic read request but does not acknowledge the aperiodic read request, thus, in this case the client has to take care of lost messages. If the request is to start a periodic read, then a series is transmitted to the client once in every cycle, the length of which is specified in the request.

6.1.2 Realization using Multiple Threads

A multi-threaded object is used as the paradigm for designing the real-time server. The structure of this object with its major components is shown in Figure 6.1. The object receives a request from the client and executes methods to accomplish the request. The server uses two kinds of threads: a mother thread and child threads. The mother thread determines the method to be executed in order to service a request and spawns a child thread to carry out the execution of the method. Each of the child threads are scheduled based on their priority, hence, in the real-time environment higher priority jobs are not adversely affected by the lower priority jobs as would have been the case if the server was single threaded.

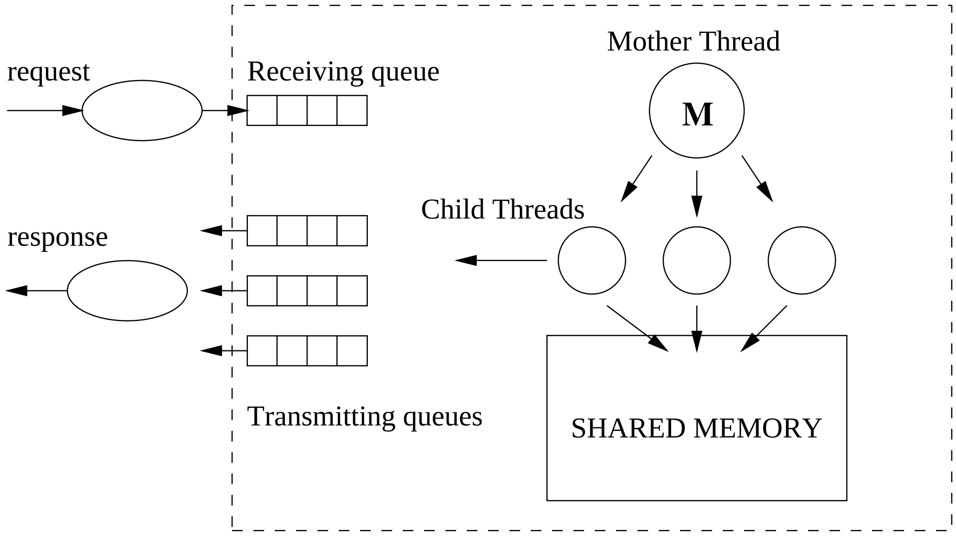


Figure 6.1: Structure of the Multi-threaded Object

The components of the real-time server are shown in Figure 6.2. The temporal data from the plant are stored on the "ring buffer", which a circular queue in main memory, which is transferred to permanent storage before being replaced by more recent data. The various tasks involved are assigned to different threads with shared memory between them. We describe the tasks of each of the component threads below.

- *Acquisition thread*: The acquisition thread periodically samples a new datum from the memory where data sampled from the plant is written, and forms a temporal datum by attaching a time stamp to the sampled datum. It registers the data on the ring buffer.

- *Saving thread*: The saving thread periodically copies a part of the temporal data on the ring buffer to permanent storage, before the part is overwritten.

- *Receiver thread*: This thread catches a message from a client and puts it in the receiving queue.

- *Mother thread*: The mother thread gets the message from the receiving queue and determines which method should be executed. It then activates a child thread to execute the determined method. For a periodic read request it creates a periodic

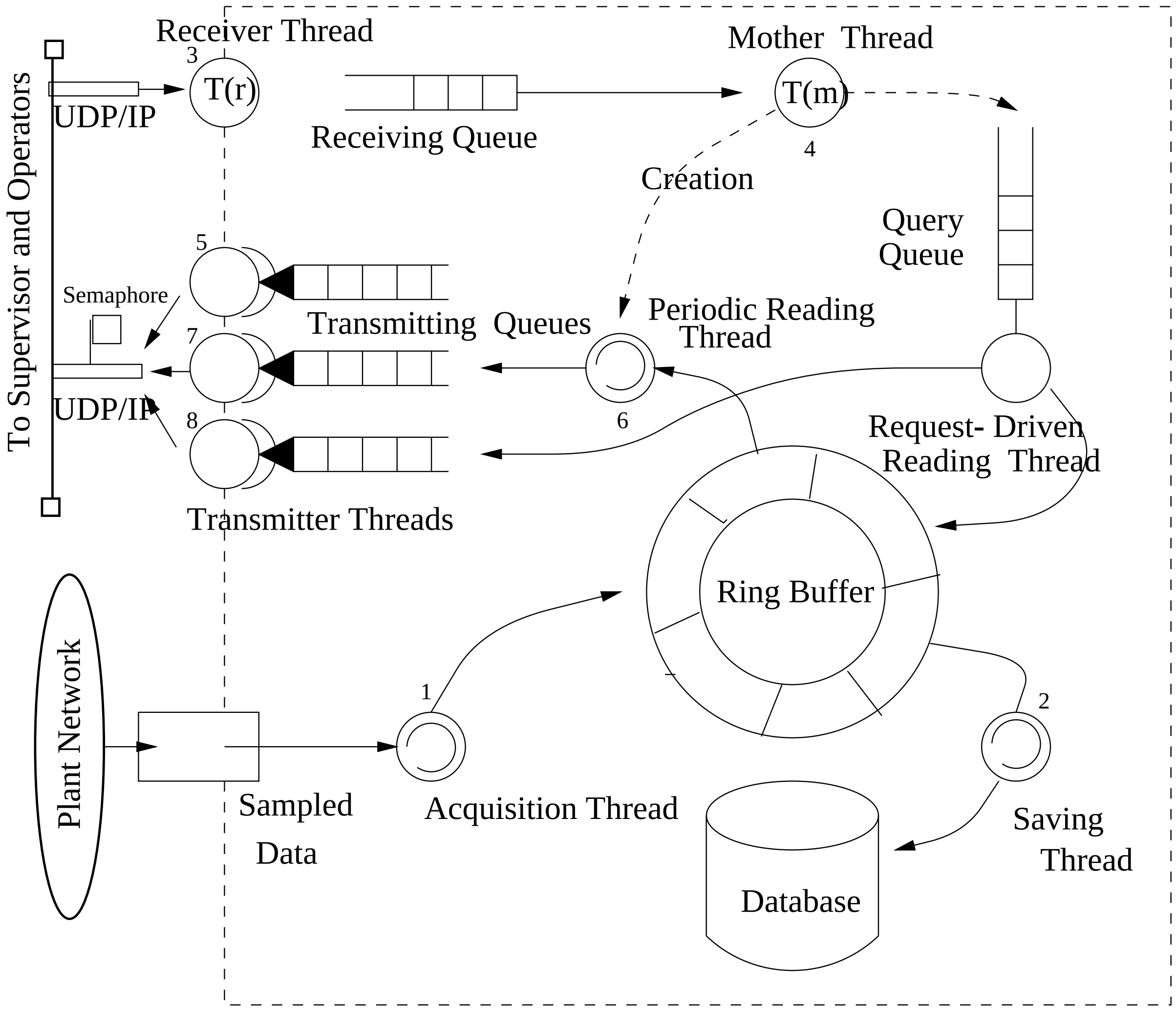


Figure 6.2: Components of the Real Time Server

thread to execute and for an aperiodic read request, it places the request in the query queue. In addition, for a periodic reading request it places the acknowledgment in the first transmission queue.

- *Periodic reading threads*: In case of periodic requests, the mother thread creates a child thread to periodically transmit the series composed of the latest temporal datum. The child thread reads the required datum from the ring buffer that always holds the data required for the longest cycle periodic read. It then composes a series and places it in the second transmission queue.

- *Request reading thread*: The request reading thread reads the messages from the query queue and for each of them it retrieves the temporal data within the period specified in the message and places them in the third transmission queue. The request driven thread runs in the background.

- *Transmission threads*: There are three transmitting queues. One contains the acknowledgments for the periodic read requests, the second holds the series composed by the periodic reading threads, and the last queue contains the series composed by the request reading thread. The transmitter threads for each queue send data and reply to clients, blocking all other transmitter threads since there is a single UDP/IP port for transmission. If only one transmitter thread is assigned to each queue, messages in the transmitting queue of higher priority cannot be sent continuously, since the transmitter thread for a lower priority queue would get the right to transmit during the time the higher priority transmitter thread gets the next message from the queue. Hence, two threads are assigned to each transmission queue.

6.1.3 Management of temporal data

Storage Structure

We can see from Figure 6.2 that a “ring buffer” is used to store the temporal data registered by the acquisition thread. The design of the real-time server presented in [Shim+93] uses a ring file to save the temporal data that was overwritten in the ring buffer on permanent storage. Since the real-time server is used in the context of a database, we use the MIDAS database as a store for old temporal data replaced in the ring buffer. Note that very old data will be overwritten with the use of a ring file, whereas in the database this data can be archived and thus never lost.

All the temporal data necessary for periodic reading should stay in the ring buffer for one cycle of the periodic read. A periodic read request may specify that only the latest temporal datum should be transmitted or that a series consisting of all the temporal data registered during the whole cycle needs to be transmitted. Thus the ring buffer must be able to hold the temporal data registered during the longest cycle of periodic reading.

Thread Priority Assignment

Each of the real-time threads are assigned a priority based on which they are scheduled. The acquisition thread is given the highest priority since it performs a hard-real-time task. The saving thread is assigned the next highest priority since it needs to write the temporal data to permanent storage before it is overwritten in the ring buffer. The receiver, mother and the periodic reading thread are assigned intermediate priorities. The transmitter threads have the next lowest priorities. In order to send acknowledgments very quickly, the first queue has the highest priority, while the last one has the lowest priority. The request driven thread runs as a background job, hence it is given a priority that is lower than the rest of the threads in the real-time server.

Request Driven Reading

The request driven reading thread tries to read the data required from the ring buffer if some or all of it is known to be available on the ring buffer before starting the read. Since the request driven thread works in the background, temporal data which it aims to read may be overwritten. A request driven thread performs an optimistic read assuming that the data will not be overwritten. In order to ensure that correct data is read, it checks the time stamp of each data read, and recognizes the point at which the data was overwritten in the ring buffer. The remaining data in the specified interval that could not be obtained from the ring buffer is read from permanent storage, which in our case is the database. Hence, a query to the database is issued to get the required data.

Mutual Exclusion and Schedulability

Since several threads access the ring buffer, each thread should exclude other threads during their access to the buffer. A locking scheme in which the whole buffer is locked during its access is prohibitive due to large blocking times since the request driven thread may read large amount of data during which the saving thread, the acquisition thread and the periodic reading thread would be blocked. Instead, a set of values called the *management tuple* are locked using a mutually exclusive lock based on priority inheritance protocol [SRL90]. According to this protocol, the priority of a thread that waits for a lock held by a lower priority thread is propagated to the lower priority thread until it finishes its current execution and releases the lock. This ensures that a thread of intermediate priority does not preempt the lock holding thread, effectively making the higher priority thread to wait for a longer time. This phenomena is called priority inversion.

The management tuple contains the index of the oldest temporal datum in the ring buffer I_s, its time stamp B_s, the index of the latest temporal datum I_e, its time stamp B_e and the index of the oldest temporal datum the saving thread has not yet written to permanent storage I_u. The acquisition thread locks the management tuple for the duration of both registration of a new temporal datum and the updation of the management tuple. The saving thread locks the management tuple during its update of I_u, before it starts copying the temporal data to permanent storage. Other threads accessing the ring buffer lock the management tuple only while they read the management tuple. The periodic thread reads the values of I_e and B_e before it reads the ring buffer. The request driven reading thread refers the values in the management tuple only once before starting the read. Since the values read by the request driven thread may not be valid after some time during the read, it has to check the time stamps of the temporal datum retrieved to ensure that the right values are being read.

The management tuple is small enough and the priority inheritance protocol ensures that a thread accessing the management tuple is blocked for the minimum amount of time. This enables us to neglect the blocking time and assume that each thread in the real-time server is independent. Threads that perform aperiodic tasks are assigned a cycle time within which only one execution of the thread is allowed, so that the rate monotonic theory developed in [LL73] can be applied to guarantee the schedulability of the real-time threads in the server.

6.2 Implementation for MIDAS-2

The real-time data server was built on the Solaris operating system that provides a real-time class for scheduling and priority based locking schemes, and supports multi-threaded programs. We can see from the MIDAS-2 architecture given in Chapter 2 that the real-time server is interfaced to the plant from which it samples the temporal data and to the MIDAS database where it stores the data with the time stamp. The graphical user interface of MIDAS is a client to the server that requests for periodic transmission of process values to be displayed through the plant interface. In addition the *process views* already provided in MIDAS-1, now obtain the process values over the specified period of time through aperiodic requests to the real-time server instead of querying the database. The implementation of real-time data server involved writing 2000 lines of C++ code.

In the following subsections we give the details of implementing the real-time server, which include the implementation of multiple real-time threads, the server's interface to the MIDAS database through the MIDAS server, the server's interface to clients, and configuring the real-time server to a specific plant.

6.2.1 Real-Time Threads

The thread library of Solaris operating system was used to implement the real-time threads. Each thread corresponds to a *light weight process (LWP)* in Solaris. In order to test the system, an emulator thread was incorporated which generates the process values for the plant at the rate of 20 milli seconds. The threads were scheduled in the real-time class provided by Solaris operating system. The real-time priorities assigned to each of the threads is given below.

Thread	Priority	Cycle (msec)
Emulator Thread	20	20
Acquisition Thread	19	20
Saving Thread	18	200
Receiving thread	17	1000
Mother Thread	16	1000
Transmitter Thread 1	15	1000
Transmitter Thread 2	13	1000
Transmitter Thread 3	12	1000
Periodic Reading	14	1000

Transmitter thread 1 transmits the acknowledgments to the periodic read requests, and transmitter threads 2 and 3 transmit the series composed by periodic and aperiodic threads respectively. Higher priority is represented by a higher number, hence the acquisition thread has the highest priority since data acquisition is a hard real-time task. The *rate monotonic priority assignment scheme* is used to assign priorities to the real-time threads. According to this scheme, tasks with higher request rates have higher priorities which ensures optimal schedulability, that is, no other fixed priority assignment scheme can schedule a task set that cannot be scheduled by this scheme.

The sample cycle times taken during testing of the system are also given in the above table. The schedulability of the system determines the rate of the acquisition task (or sampling rate) and the number of periodic threads that can be supported. Once the sampling rate is fixed the cycle times for the the saving thread and the periodic threads are chosen relative to it. The receiver, transmitter and the mother threads are not inherently periodic but are activated only when there is a message to receive and send. The cycle times for these threads give the minimum time interval between the arrival of consecutive requests. This time interval T_{msg} is set such that

$$T(\text{saving thread}) \leq \mathrm{T_{msg}} \leq \mathrm{T(periodic\ thread)},$$

where $T(t)$ gives the cycle time of the thread t.

Mutually exclusive access to the ring buffer by the threads is ensured by each thread trying to acquire a lock on the management tuple. The Solaris operating system provides a *mutex* or mutually exclusive lock for this purpose. For the threads scheduled in the *real-time class* the operating system implements the priority inheritance protocol for the threads blocked on the *mutex*. When an thread of higher priority blocks on a *mutex* held by a lower priority thread, it sets the priority of the lower priority thread equal to that of the blocked thread until the *mutex* is released. This does not allow delaying of the lower priority process due to preemption by a thread of intermediate priority.

6.2.2 Client Interface

It was mentioned in the design of the real-time server that the UDP/IP protocol that is a connection-less and unreliable protocol is used for communication between the clients and the server. We use the UNIX network features for the client-server communications. Any client can send a message to the server on a predefined port provided the server is running on the specified host. The server receives messages from the client by listening on the port. When a message is received, the address of the client that sent the message is also returned to the server. This address is used for further communication with the client, that is, to send acknowledgments and to transmit series of temporal data. Since, there are multiple transmission threads which can transmit over a single port, access to the port for transmission is made exclusive by using a semaphore. Since a message is just a sequence of bits, the format for the requests, the acknowledgments and the series of temporal data should be known to both the client and the server. These formats are given in a common "C++" header file that can be used by the client program.

6.2.3 Configuring the Real-Time server

As mentioned in the design of the server, the plant interface to the server is through a shared memory into which the data sampled from the plant is written. The shared memory is implemented using the Unix *mmap* command. Initially, when the server is started the number and sizes of the characteristic variables has to be known so that they can be read from this shared memory. In addition the name of the file used by *mmap* has to communicated to the server. Since the server writes the process values to the database, the name of the sensor (sensor_id) to which each characteristic variable corresponds has to be known to the server. The name of the mmap file and the sizes and names of the plant variables and the required sampling rate should be communicated to the server through a configuration file that is read by the server at the start of its execution.

6.2.4 Interface to MIDAS server

In MIDAS-2, the real-time server stores the temporal data in the database. The saving thread inserts each of the process values with their respective sensor names and time stamp at which they were sampled into the database using the routines in the MIDAS server for interfacing with the database. As already mentioned, the sensor name corresponding to each process value sampled is given in the configuration file. The request driven thread tries to read the data required, from the ring buffer if it has not been overwritten. The data that is no longer present in the ring buffer has to be acquired by querying the database for the required range of time stamps.

Since Illustra allows archival of old data, the table containing the temporal data could be occasionally 'vacuumed', that is, the present contents are moved to another table on a backup disk which is rarely used. Queries are unchanged and can refer to data in the archival tables.

Chapter 7

Semiconductor Plant Example

In MIDAS-1, the initial prototype of the MIDAS system was customized to a Coffee Plant example which was a simplified form of FMS. We have adapted MIDAS-2 to a semiconductor device manufacturing plant example that is a simplified version of a typical semiconductor plant. This plant is a more complex example, since the machines used in semiconductor device manufacture are automated to a large extent and can be programmed to perform any one of a set of operations. We assume that the equipment is grouped into cells, where each cell has a cell controller and that a plant controller is responsible for the overall control of the plant. This plant uses the MIDAS database for data management and for informing the operator of exceptional events. The MIDAS-2 server provides decision support for scheduling the products and the RTDS provides real-time access to process values generated from the sensors of the plant.

7.1 Description of the Plant

This example is a simplified version of the semiconductor fabrication process for devices. The equipment used in the example are commonly used in the semiconductor manufacturing industry [GB86]. However, we restrict our example to the manufacture of simple devices like transistors. The extension to Integrated circuit fabrication requires the repetition of steps shown in the example many number of times. In addition, each step in the example may correspond to a sequence of operations that include cleaning, heating, etc. This simplified model was derived from descriptions given in [GB86, BG89, Chen+88]. The layout of the example semiconductor plant given in Figure 7.1 shows the machines and the different possible routings in the plant.

The semiconductor devices are manufactured in groups, where each group of devices comes from a single "wafer". A wafer is a circular piece of silicon, that is carefully cut from a pure silicon crystal. The silicon crystals are grown from purified silicon dioxide (sand). A wafer goes through a set of machines that dope it appropriately to make a group of similar devices. Doping involves injecting ions of a particular charge (positive or negative) into the pure silicon. Each such device on the wafer is called a "die". At the end of the doping process, the metallic contacts between the doped areas are made, after which the dies are separated and packaged. The various processing steps involved in the manufacture of devices in the example plant are described below.

- *Crystal Growth*

  This step involves growing an individual silicon crystal using pure silicon obtained from silicon dioxide.

- *Wafer Manufacture*

  The silicon crystal is ground perfectly round and sawed to make wafers. The crystal growth machine and the wafer manufacturing machine are grouped into a *wafer preparation cell.*

The next step is wafer fabrication, in which a sequence of oxidation, ion implantation, photolithography, etching, resist strip, epitaxy and metallization processes are done. The fabrication unit is a separate cell with two flexible transporters within the cell. The exact sequence in which these operations are done depends on the product type and may involve going through each of the above mentioned processing steps more than once.

- *Oxidation of Silicon*

  In this step a layer of silicon dioxide is grown on the wafer. It is used as preferential masking layer during the fabrication sequence.

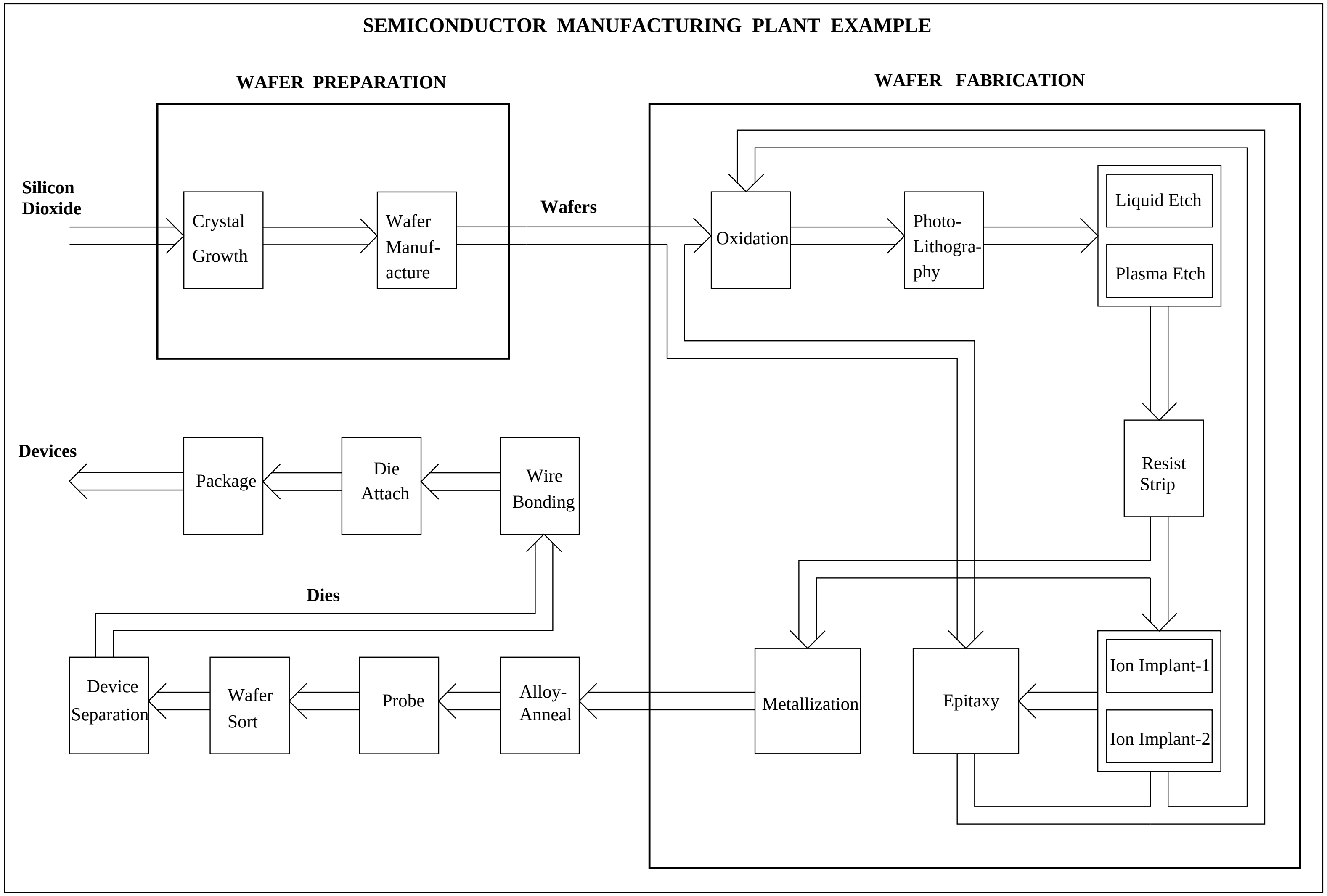


Figure 7.1: Plant Layout of Semiconductor Example

- *Photolithography*

  In this step, the wafer coated with a special material called "resist" is exposed to the light so that a mask is developed. The mask determines the areas of the wafer that have to be exposed during a later processing step which may be ion implantation, epitaxy or metallization. The mask to be developed on the wafer is different for each step of processing the wafer and for each type of device being fabricated. Hence, the appropriate mask is loaded by the unit from the database.

- *Etching*

  In this step, the layer of silicon dioxide not protected by the resist is removed by a process called etching. The plant has plasma and liquid etchers which can be used for the this purpose.

- *Resist-strip*

  After the etching process, the resist on the wafer is removed. The sequence of operations followed for "mask preparation" are lithography step, etching of the masking layer and resist-strip. The mask developed after this sequence is checked by a defect detector. If the mask fails to match, the work is scrapped and the wafer is reworked.

- *Epitaxy*

  Some semiconductor devices have a deposition of single crystal layer of silicon as an extension to the crystal structure of the substrate. This layer called the epitaxial layer, is formed in the epitaxy unit. This unit has sensors that continuously monitor the temperature and the velocity of the gases flowing in the chamber. The controller of the unit automatically adjusts these parameters to the appropriate values when the unit is under operation using the feedback from the sensors. The values of these sensors are sampled by the plant controller.

- *Ion Implantation*

  The plant has two Ion implanters which are similar in operation. The doping of unmasked areas of the wafer with ions of a particular dope type is done during ion implantation. The ion implanter has manual controls for adjusting the dope depth, dope type (p or n) and the dope strength.

- *Metallization*

  The electrical contacts for the devices are made during this process.

After device fabrication, the sequence of steps the wafers go through to finally become devices is given next.

- *Alloy-Annealing*

  This step forms a low resistance contact between the aluminium layer that was formed during metallization and the silicon.

- *Probing*

  The performance of devices from different areas of the wafer are tested and if the wafer is suspected to contain less than an economical number of good devices, it is discarded.

- *Wafer Sort*

  All the devices on a wafer are probed and the non-functional devices are marked.

- *Device Separation*

  At this stage the wafer is cut to separate the individual devices, and the marked devices are discarded.

- *Die-Attach*

  The backside of the die is attached to a layer of metal.

- *Wire Bonding*

  The bonding pads of the device are connected to the package.

- *Packaging*

  The devices are covered with metal or ceramic to protect them from the environment.

In the plant, three types of products namely, bipolar *npn* transistor, bipolar *pnp* transistor and the *cmos* transistor are manufactured. The sequence of operations that the wafer goes through during wafer fabrication is different for each of the products. The wafer fabrication step involves repeated mask preparations followed by either an ion implantation, epitaxy or metallization. All the steps excluding the wafer fabrication step are followed in the sequence given above.

The plant controller takes the schedule prepared from the database and initiates the production. It routes the lots through the appropriate machines by sending *START* and *SETUP* commands to the machines and *MOVE*, *LOAD* and *UNLOAD* commands to the transporters. It receives the status information about the current location of every transporter that is moving, states of the machines, and process values from the sensors, which it stores in the database, using the MIDAS server routines.

This example has a number of fully and semi-automated machines, fixed and flexible-motion transporters, sensors and defect detectors, complicated routing for a variety of product types, etc. that make it suitable to demonstrate how MIDAS-2 handles the control and data management for a flexible manufacturing plant of non-trivial size. The Real-time Data Server is used to acquire the sensor values for this plant in real time. The Lagrangian and the heuristic scheduler were used to schedule the orders for the semiconductor products.

7.2 Implementation of the semiconductor plant model

7.2.1 Implementing the Data Model

A subset of the objects and associations defined in the generic object model of MIDAS-1 was sufficient to model the "static" structure of the entities in the semiconductor plant. The control aspects of the semiconductor plant operation are captured by a subset of the events and their effects on the states of objects as given in the dynamic model for the manufacturing domain.

The important classes and associations used in implementing the example plant are given below.

- *Machines*

  All the machines in the example plant are instances of this class. The Photolithography unit is program driven, since it can be programmed to prepare any required mask. The Ion Implanter has manual controls that can program it to implant ions of the type specified to the required depth and dosage. Each of these program driven machines are modeled as an aggregation of their respective controllers which are instances of *Controller_unit* and *manual_controls* classes. The plant controller is an instance of the *Supervisory_control* class.

- *Transporter*

  Two flexible-motion transporters are instantiated, each of which can be programmed to move between a set of predefined points. The crystal growth and wafer preparation machines and all the machines starting from the alloy-annealing unit to the packaging unit are connected by fixed motion transporters. These transporters move along one fixed route.

- *Operations*

  Each of the given processing steps are separate operations. In addition, the various transporter operations like loading, unloading and transporting are also included in the operations class.

- *Workpiece*

  The raw material, products and the intermediate parts that are produced during the processing of raw material into product form the instances of workpiece class. For each product type, the wafer or die that results after each processing step is a workpiece with different properties.

- *Lots*

  For every lot, the information about the time spent at each processing unit is stored in this class. Lots are generated when the plant controller initiates the production of a particular lot. Also, when the units are to be reworked they are made into new lots that are routed independently.

- *Products and Orders*

  The products class has information about the final products, namely, bipolar *npn*, bipolar *pnp*, and *cmos* transistors. The orders for each product include the quantity of product required, the dispatch date and the priority of the order.

- *Sensors*

  The sensors class includes the defect detector for checking the masks developed after the lithography step, the optical sensor in the epitaxy unit to measure the temperature of the unit, and the sensor that indicates the gas flow velocity in the epitaxy unit.

- *Transient_Store*

  Each machine has an input buffer to hold the lots waiting to be processed and an output buffer where it places the lots that have completed processing and waiting to be transported to the next machine.

The above classes model the various entities in the plant. The association between the entities shows their interrelations.

- *Process_Sheet*

  It gives all the possible routes that a lot can take and the time required to process each part on each machine, which is useful during scheduling.

- *Route_Card*

  It gives the start and end times of each operation that each lot goes through during production.

- *Sub_Lot*

  It stores the relation between the parent lot and its sublots.

- *Sub-Orders*

  The orders for the product are divided into suborders, where each suborder corresponds to a single lot of products.

- *Workpiece_Subpart*

  The relation between the various parts that make up the product, that is, the bill of materials information is stored as an aggregation relation between the workpieces.

7.2.2 Incorporating Triggers

The active database features supported by Illustra engine were used to inform the operator about transporter and machine malfunctions. In addition, whenever the number of defective "dies" in a wafer is above a threshold the operator is informed that the yield has deteriorated. Rules are incorporated in the database which check the state of the transporter whenever the transporter class is updated, and if the state is being updated to *FAIL*, an alerter corresponding to the particular transporter is fired. An example of a rule used to fire an alerter when the flexible transporter *FT1* fails is given in Illustra SQL as below:

```
create rule fail_signal
on update to flexible_motion
where new.state = 'FAIL' and new.vehicle_id = 'FT1'
do
alert FT1_fail;
```

The rule specifies that when the state of the transporter "FT1" changes to *FAIL* the alerter "FT1_fail" is fired. At present we have incorporated rules to inform the failure of machines and flexible transporters. Similar rules are inserted to inform the operator of very low yields.

Each of the alerters is associated with a callback. Whenever an alerter is fired, the callback associated with the alerter is executed. These callbacks are routines that inform the operator of the specific events through an audio interface.

7.2.3 Operation of the Plant

The schedule for the manufacture of products is prepared off-line and stored in the database from which the plant controller retrieves it before starting the operation of the plant. As lots are routed through the plant, the start and end times of each operation the lot goes through is stored in the database. The status of the machines, transporters, transient_stores and sensor values are continually reflected in the MIDAS database. The values of the sensors that indicate the temperature and gas velocity in the "epitaxy" unit will be sampled every second and sent to the plant interface that writes these values in the shared memory, from which the real-time server reads the values. The defect detectors note the number of defective units out of the total units processed. Whenever the yield is very low, an alerter is fired which automatically calls the interface routines that inform the operator. The flexible transporters, and the program-driven machines are loaded with the appropriate programs before executing the required operation. This loading time is assumed to be negligible relative to manufacturing operations.

7.2.4 Implementation Details

A simulator for the semiconductor plant example was developed to simulate the events generated by the equipment controllers. Events like "processing over", "loading done" are sent by the machine and transporter controllers to the MIDAS server. The server has control routines that enable it to perform the functions of the plant controller. These routines receive events from the plant and send commands to run the plant. The status data generated during the plant operation are recorded in the database by the MIDAS server. The plant interface samples the values of the sensors and communicates it to the real-time server through shared memory. The real-time server is informed about the names of the sensors and the format of the plant information when it is started up. The values of sensors are recorded in the database by the real-time server, from which they can be retrieved for fault, performance or defect analysis.

About 36 classes and associations out of the 150 classes in the MIDAS Object model were used for modeling the semiconductor plant example. Almost equal number of Illustra functions were written. Rules were written for the objects in the flexible_motion transporter class, the machines class and the defect_detector class. The simulator, the plant interface to the real-time server and control routines in MIDAS server were written in 2500 lines of C++ code.

7.3 Customizing MIDAS

The MIDAS system was adapted to two example plants, namely the Coffee Plant and the Semiconductor plant. The Coffee Plant was a simplified example with limited routing and the machines and transporters had predefined operations. The adaptation to Coffee plant was done as part of MIDAS-1. The semiconductor plant example had flexible machines and transporters that could perform a number of operations. The same set of machines can be used to produce new types of products by defining a new route for the product. It can also be seen that the plant can easily adapt to changes in product mix.

Customizing MIDAS to a specific plant involves incorporating the existing control routines of the plant controller into the MIDAS server, and configuring the real-time server appropriately. The server reads the configuration of the plant at run-time, hence the code need not be recompiled to be used for a different plant. The working of the schedulers is independent of the plant details since all the information required for the scheduling is obtained from the database and the resulting schedule is written back to the database. In addition methods like *simulated annealing* can be incorporated into the system for preparing schedules for plants where the Lagrangian scheduler cannot be applied. Some of the control software could be converted into rules and added to the database. The dynamic model helps in getting a clear understanding of the control structure of the plant for incorporating the required rules and triggers.

Many automated plants have an already existing relational database to store part programs, status information, etc. Switching to MIDAS for such plants is easy, since the data in the relational database could be mapped to Illustra tables. However, if the existing database uses any other data model, then the various entities in the plant have to be matched to the corresponding objects in the MIDAS object model. Any additional classes required can be easily added.

Many automated manufacturing plants already have their own passive database systems to store information, tools for Materials Requirement Planning (MRP), and expert systems for plant monitoring and exception handling. These tools can be incorporated into MIDAS system and need not be discarded. Customizing MIDAS system for information management of a specific plant may involve a few months of effort to incorporate the plant's structural details and control.

Chapter 8

Multimedia User Interface

A multimedia user interface was developed to provide the operator with a continuously updated view of the plant. The interface interacts only with the database and does not communicate directly with the controllers of the plant. The use of an integrated database system and controller simplifies the task of developing an interface for a particular plant. The details of the user interface provided for the semiconductor plant are given in this chapter.

8.1 The Semiconductor Plant Interface

The interface to the semiconductor plant is mostly built using X and Motif [Moti94]. In addition to the *visual* interface, there is an *audio* interface that is used to notify the user of exceptional events like equipment failures and other plant specific events. The various kinds of information provided by the interface to the user and their method of presentation are detailed in the following subsections.

Structural Information

The machines, transporters, and transient stores in the plant are displayed in the interface. The structural information about each of the above objects can be queried by clicking on the corresponding icon. For example, when the user clicks on any of the machine icons the machine_id, type of the machine, the simple cell it belongs to are displayed. If the machine is program driven, the information about the controllers and manual controls it contains is also displayed. In addition, the details of all the lots under processing can be obtained by clicking on the lot icon that is displayed for every lot under processing. Lot details include the lot_id, the present workpiece code, number of lot units, and the final product code.

Status Information

The various objects in the plant continuously change their states. The plant display is frequently updated to reflect the current status of the plant without explicit querying by the operator.

- A machine could be either in *IDLE*, *SETUP*, *PROCESSING* or *DOWN* state. As soon as a machine changes its state, the background color of the icon representing the machine changes to indicate the new state of the machine. When the machine is in the *PROCESSING* state, the name of the current lot under processing is also displayed.

- Fixed motion transporters can be in either *IDLE* or *MOVING* state, while the flexible motion transporters can also be in *LOAD* or *UNLOAD* states. The different routes along which the transporters can move are shown in the display. When the transporter is in motion its location is updated at a certain frequency in the database. The current location is reflected in the plant display by moving the transporter icon to the particular offset on the particular route.

- The current contents of input and output buffers of every machine are displayed. If the buffer contains a lot, its name is displayed in the buffer icon. If the buffer is empty then the label reads 'NIL'.

- The values of the sensors have to be immediately displayed to the operator. The sensors are represented as scales, with an indicator that moves along the scale to display the current value of the parameter that the sensor is measuring.

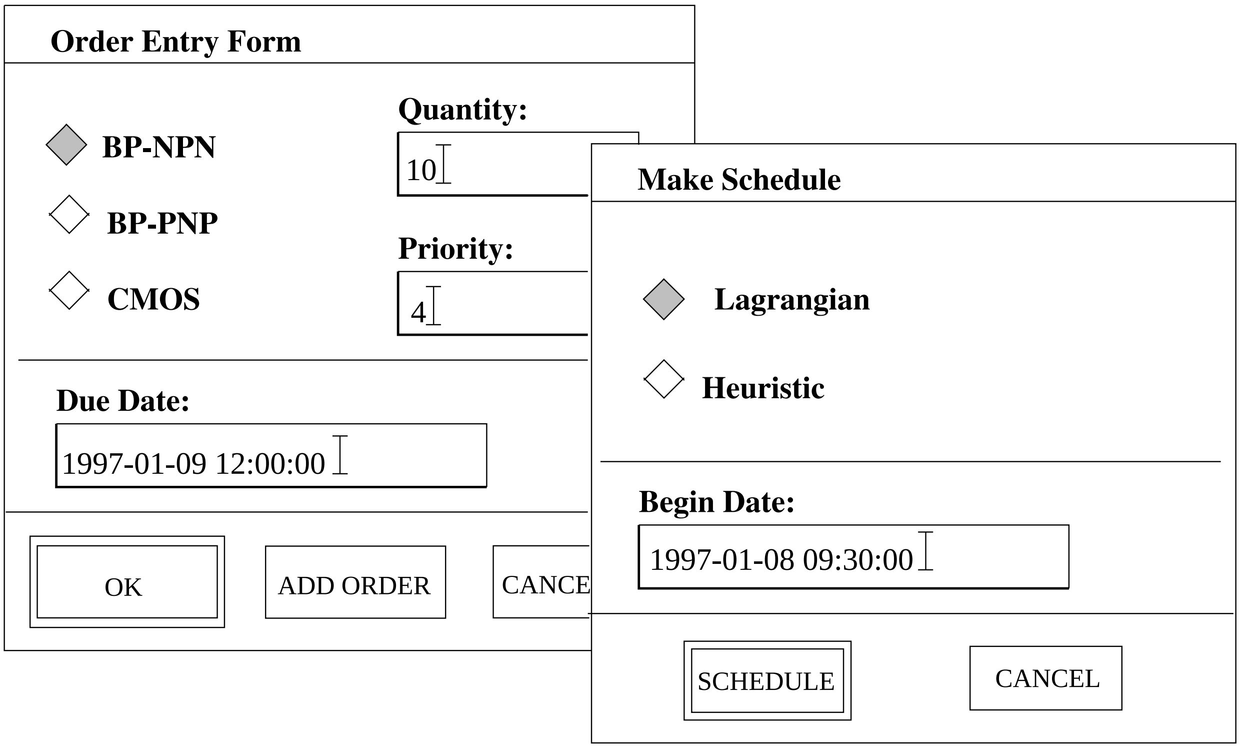


Figure 8.1: Order Entry and Scheduling Forms

Order Entry and Scheduling

The plant interface provides forms for order entry. The order information is stored in the database. In addition, the set of current orders can be scheduled by selecting the scheduler and giving the start data of scheduling through another form. A set of orders can be entered through the form and subsequently one of the schedulers can be invoked to prepare the schedule as mentioned in Chapter 5. The structure of the forms for order entry and scheduling are shown in Figure 8.1.

Notifying Exceptions

We have seen in the last chapter that triggers are included to notify the application about exceptional conditions. When a notification is received, a window is popped up which displays an appropriate message to inform the operator about the exceptional event. This is accompanied by an audio message that informs the operator of the particular event. The audio interface is helpful in drawing the attention of the operator to these exceptional events. In the semiconductor plant, the events notified are failure of machines and flexible_transporters, and information of very low yield.

Incorporating Views

The view generation routines developed in MIDAS-1 are incorporated in the interface of the semiconductor plant. The routines were incorporated with very little changes, since they read information from the database and generate the display irrespective of the specific contents. The plant view shows all the lots that were processed within a time interval and the time slot in which they were being processed by each machine. The process view shows the temperature and the velocity of gases in the epitaxy unit while each lot was being processed.

In addition, a control panel provides the operator with an interface to directly control the various equipment in the plant. For example, the operator can stop, start or change the configuration of a machine by clicking on the corresponding control button which results in sending the appropriate control signal to the machine controller.

8.2 Implementation Details

The interface waits for events from the user like any other X application. In addition, it waits for triggers from the database that inform that the status of some objects have changed or that an exceptional condition has arisen. Sensor values are requested from the real-time server which then periodically transmits the values to the interface program. The interface program also waits for the transmission of these sensor values. Thus, the program consists of multiple interacting threads. One thread waits for events from the user in the 'XtMainLoop' of X and two other threads each wait for database triggers and receipt of information from the real-time server. Whenever the display has to be updated to reflect the status of equipment or the values of the sensors, the corresponding display routine is added to the list of routines that X executes when there are no events from the user.

Exceptional events are notified by triggers that are invoked by the rules that check for these conditions. When such a trigger is received, it indicates either a machine or transporter failure, or that yield has gone below the economic level. Correspondingly, an appropriate message is played on the audio device to draw the attention of the operator, along with the visual message.

On receiving the trigger, the application has to retrieve the particular status information that was updated and update the display accordingly. The exact implementation involves creating rules that fire alerters when there is a change in the status information of the particular object. Since alerters cannot convey data values, the database has to be read to get the updated status. A callback is associated with the "alerter_fired" event defined in Illustra, which will be invoked whenever any alerter fires. The appropriate display routine that updates the status of the object is called for each alert signal.

Chapter 9

Conclusions

9.1 Summary

The MIDAS project was aimed at building an integrated database system with embedded control for flexible manufacturing systems. MIDAS-1 dealt with the issues of classifying and analyzing the data in the manufacturing domain, developing the object model, integrating active mechanisms and providing simple decision support. While it was a good first step, it had some limitations. The nature of control flow between the various objects in the manufacturing plant was not obvious from the object model developed in MIDAS-1. Efficient algorithms based on mathematical techniques that can be applied to scheduling problems of practical size and complexity have been recently developed that give significantly better results than the heuristic approach being used for scheduling in MIDAS-1. The manufacturing plant example used to demonstrate the data management and control aspects of MIDAS was a simplified form of FMS.

These limitations were overcome in MIDAS-2 which was developed during this project. In MIDAS-2, the requirement of providing real-time response and effective decision support, and modeling the control aspects of the manufacturing system to make it better tuned to the task of supporting integrated manufacturing systems has been addressed.

In this project, we have designed the *dynamic model* that shows the events generated by the various objects in the manufacturing plant and the effect of these events on the states of the objects. The dynamic model gives an augmented view of the manufacturing domain. A scheduler based on *Lagrangian Relaxation Technique* which had been developed to produce near-optimal schedules was incorporated. This scheduler generates a schedule whose cost is within a small gap of the minimum cost. A *real-time data server* has been incorporated, which acquires the temporal data from the plant and registers it with a time stamp, and sends the data to requesting clients in real-time. The real-time server is interfaced to the MIDAS database where it stores the temporal data. The MIDAS-2 system was customized for a semiconductor device manufacturing plant example, that was a representative example of an FMS with flexible automated machines, complicated routing and a variety of products. About 36 classes and associations out of the 150 classes in the MIDAS Object model were used for modeling the semiconductor plant example which had 16 processing units, and an equal number of fixed and flexible motion transporters. A simulator for the plant was written to test the operation of the final system. Incorporating the advanced features and simulating the semiconductor plant example involved writing 7000 lines of C++ code.

A Multimedia User Interface was provided for the example plant, which reflects the latest state of the plant through a graphical interface and informs the operator of exceptional events through an audio interface. Triggers were incorporated for the example semiconductor plant, that allowed the database to respond to certain events by sending signals. These signals are presently used to inform the operators who take appropriate actions in case of exceptional events. In turn, the plant controller can be enabled to handle these conditions with the aid of decision making software and expert systems.

9.2 Suggested Extensions

The MIDAS system can be extended to incorporate extra functionalities that would increase its applicability to a wider variety of manufacturing systems.

- Existing manufacturing systems have their data distributed in small databases each having information concerning a sub-system which is stored in the vendor's proprietary formats and structures. A uniform interface to this data could be provided to the operator independent of the formats of the individual databases by using the Multi-Database approach [BGS92]. This involves providing gateways from MIDAS to the various sub-systems in the manufacturing system.

- The scheduler based on Lagrangian Relaxation incorporated in MIDAS can be used to generate data that can be used for answering "what if" questions without the need for expensive simulations. In addition, the schedule generated could be used to get a new schedule with minor changes to the workload or input parameters like machine capacities, within a fraction of the time it takes to generate a completely new schedule. An example of workload change is the addition of a new job. Support for the above features could be incorporated by modifying the present scheduler.

- Activities in the manufacturing system other than shop floor activities are not presently dealt with by the MIDAS system. It could encompass the business and planning activities like capacity planning, financial management and master production planning.

Bibliography

[Beeb83] W. Beeby, "The Heart of Integration: A Sound Database", *IEEE Spectrum*, Vol.20, No. 5, May 1983.

[BG89] X. Bai and S. Gershwin, "A Manufacturing Scheduler's Perspective on Semiconductor Fabrication", *VLSI Publications, MIT*, April 1989.

[BGS92] Y. Breitbart, H. Gracia-Molina and A. Silberschatz, "Overview of Multidatabase Transaction Management", *VLDB Journal*, Vol. 1, No. 2, October 1992.

[Bhuv96] R. Bhuvaneshwari, "Design and implementation of MIDAS-A Manufacturing Information DAtabase System", *Project report, Indian Institute of science*, June 1996.

[Booc94] G. Booch, *Object Oriented Analysis and Design with Applications, 2nd Edition*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[Came+75] P. Camerini et al., "On Improving Relaxation Methods by Modified Gradient Techniques", *Mathematical Programming Study 3*, 1975.

[CB90] D. Chan and D. Bedworth, "Design of a Scheduling System for Flexible Manufacturing Cells", *International Journal of Production Research*, Vol. 28, No. 11, November 1990.

[CC91] I. Chen and C. Chung, "Effects of Loading and Routeing Decisions on Performance of Flexible Manufacturing Systems", *International Journal of Production Research*, Vol. 29, No. 11, November 1991.

[Chen+88] H. Chen et al., "Empirical Evaluation of a Queueing Network Model for Semiconductor Wafer Fabrication", *Operations Research*, Vol. 36, No. 2, March-April 1988.

[CL94] C. Czerwinski and P. Luh, "Scheduling Products with Bills of Materials Using an Improved Lagrangian Relaxation Technique", *IEEE Transactions on Robotics and Automation*, Vol. 10, No. 2, April 1994.

[DJ90] R. Dayal and A. Joshi, "An Approach to Computer Aided Scheduling", *Proceedings of the 14th All India Machine Tool Design and Research Conference*, December 1990.

[ElRe+95] H. El-Rewini et al., "Object-Technology: A Virtual Roundtable", *IEEE Computer*, Vol. 28, No. 10, October 1995.

[EN94] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1994.

[Eykh+92] J. Eykholt et al., "Beyond Multiprocessing-Multithreading the SunOS Kernel", *Sun Technical Bulletin*, August 1992.

[Fish73] M. Fisher, "Optimal Solution of Scheduling Problems using Lagrangian Multipliers, Part I", *Operations Research*, Vol. 21, 1973.

[Fish81] M. Fisher, "Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science*, Vol. 27, No. 1, January 1981.

[GB86] P. Gise and R. Blanchard, *Modern Semiconductor Fabrication Technology*, Prentice-Hall, 1986.

[GR93] J. Gray and A. Reuter, *Transaction Processing: Concepts and Technology*, Morgan Kaufmann Publishers, Inc., 1993.

[Groo87] M. Groover, *Automation, Production Systems and Computer Integrated Manufacturing*, Prentice-Hall, 1987.

[Harh+94] G. Harhalakis et al, "Implementation of Rule-Based Information Systems for Integrated Manufacturing", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No.6, December 1994.

[HB90] D. Hearn and P. Baker, "Computer Graphics", Prentice-Hall of India, 1990.

[Hoit+90] D. Hoitomt et al., "Schedule Generation and Reconfiguration for Parallel Machines", *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 6, December 1990.

[HS89] M. Hardwick and D. Spooner, "The ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 2, June 1989.

[Hutc+91] J. Hutchison et al., "Scheduling Approaches for Random Job Shop Flexible Manufacturing Systems", *International Journal of Production Research*, Vol. 29, No. 5, May 1991.

[IAPI95] *Illustra Application Programming Interface Guide*, Illustra Server Release 3.2, Illustra Information Technologies, Inc., 1995.

[IDAG95] *Illustra Direct Access Users Guide*, Illustra Server Release 3.2, Illustra Information Technologies, Inc., 1995.

[IUG95] *Illustra Users Guide*, Illustra Server Release 3.2, Illustra Information Technologies, Inc., 1995.

[KA90] S. Kim and H. Ahn, "Convergence Properties of the Modified Subgradient Method of Camerini et al.", *Naval Research Logistics*, Vol. 37, 1990.

[Kore83] Y. Koren, *Computer Control of Manufacturing Systems*, McGraw-Hill, 1983.

[KR95] R. Kumar and T. Roy, Unpublished manuscript, Information Technology Services of Tata Steel, Jamshedpur, 1995.

[KS91] H. Korth and A. Silberschatz, *Database System Concepts*, Second Edition, McGraw-Hill, 1991.

[KSZ93] S. Khanna, M. Sebree and J. Zolnowsky, "Realtime Scheduling in SunOS 5.0", *Sun Technical Bulletin*, January 1993.

[LL73] L. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM*, Vol. 20, No. 1, 1973.

[LRB77] J. Lenstra, A. Rinnooy and P. Bruckner, "Complexity of Machine Scheduling Problems", *Annals of Discrete Mathematics*, Vol. 7, 1977.

[Lugg91] W. Luggen, *Flexible Manufacturing Cells and Systems*, Prentice-Hall, 1991.

[Moti94] *Motif Programming Manual*, O'Reilly & Associates, Inc., 1994.

[SS94] *Multithreaded Programming Guide*, SunSoft Inc., August 1994.

[Nath88] J. Nathan, "MRP in CIM for Selected Products", *CAD/CAM Robotics and Factories of the Future*, Vol. 2, 1988.

[Powe87] J. Powers Jr., *Computer-Automated Manufacturing*, McGraw-Hill, 1987.

[Rumb+91] J. Rumbaugh et al, *Object Oriented Modeling and Design*, Prentice-Hall, 1991.

[Rhod85] S. Rhodes, "The Development of Integrated Control Systems for Flexible Manufacturing Systems", *AUTOFACT 85, Conference Proceedings*, Nov 1985.

[Sepe87] M. Sepehri, "Integrated Data Base for Computer-Integrated Manufacturing", *IEEE Circuits and Devices Magazine*, March 1987.

[Shap79] J. Shapiro, "A Survey of Lagrangian Techniques for Discrete Optimization", *Annals of Discrete Mathematics*, Vol. 5, 1979.

[Shim+93] H. Shimakawa et al., "Aquisition and Service of Temporal Data for Real-Time Plant Monitoring", *Proceedings of the Real-Time Systems Symposium*, December 1993.

[SRL90] L. Sha, R. Rajkumar and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Transactions on Computers*, Vol. 39, No. 9, September 1990.

[Stro91] B. Stroustrop, *The C++ Programming Language, 2nd edition*, Addison-Wesley, 1991.

[Taka+95] H. Takada et al, "A Data Model and Views for Manufacturing Management Databases", Unpublished manuscript, IESL, Mitsubishi Electric Corporation, Japan, 1995.

[Taka+96] H. Takada et al, "Production Information Management for Batch Manufacturing Plants Based on ECA Mechanism and View Generation", Unpublished manuscript, IESL, Mitsubishi Electric Corporation, Japan, 1996.

[VN92] N. Viswanadham and Y. Narahari, *Performance Modeling of Automated Manufacturing Systems*, Prentice-Hall of India, 1994.

[WJ90] R. Wirfs-Brock and R. Johnson, "Surveying Current Research in Object-Oriented Design", *Communications of the ACM*, Vol. 33, No. 9, September 1990.

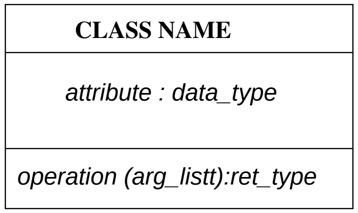
[Youn90] D. Young, *The X Window System*, Prentice-Hall, 1990.

Appendix A

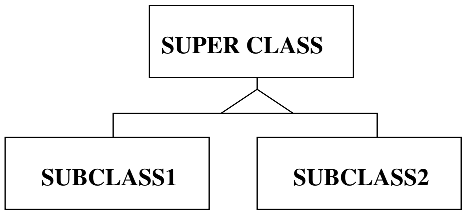
Rumbaugh Notation

Object Model Notation

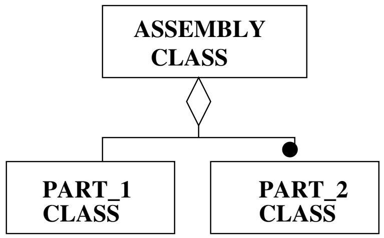
Class :



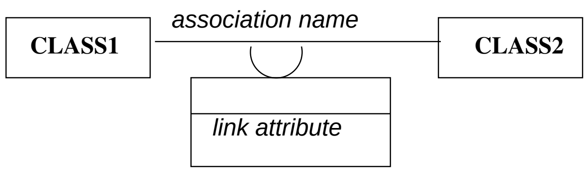
Generalization(Inheritance)



Aggregation

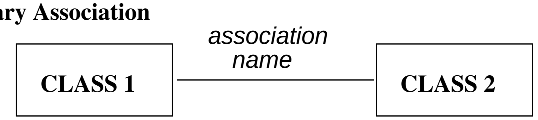


Link

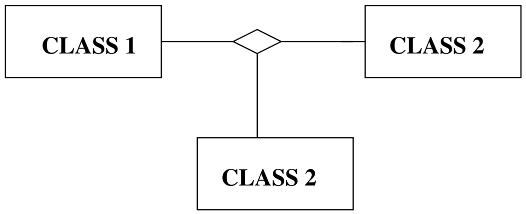


Association

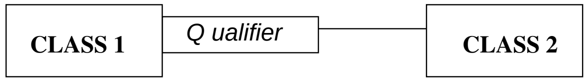
Binary Association



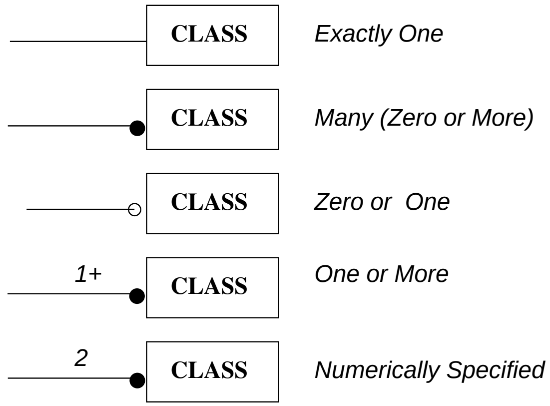
Ternary Association



Qualifiied Association



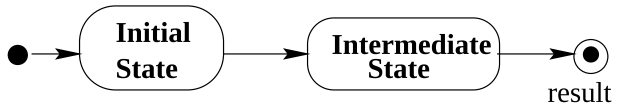
Multiplicity of Associations



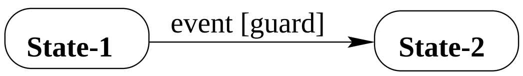
Dynamic Model Notation

Event causes Transition between states:

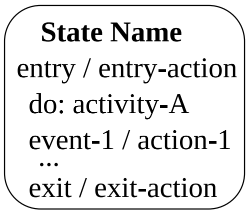
Initial and Final States:



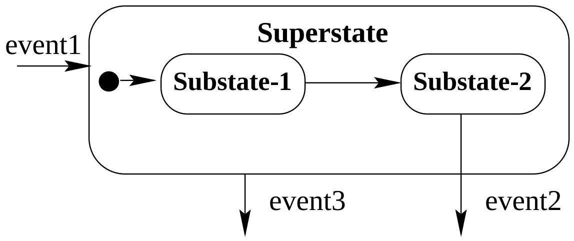
Guarded Transitions:



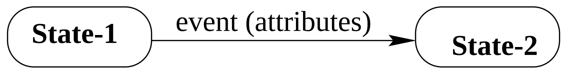
Actiona and Activity while in the State:



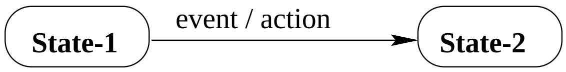
State Generalization (Nesting):



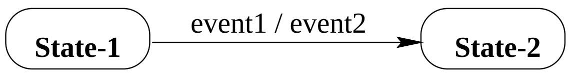
Event with Attribute:



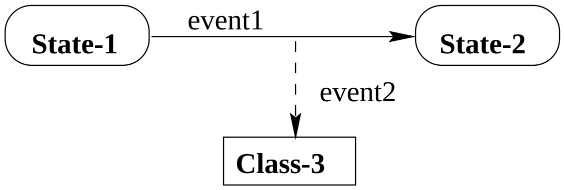
Action on a Transition:



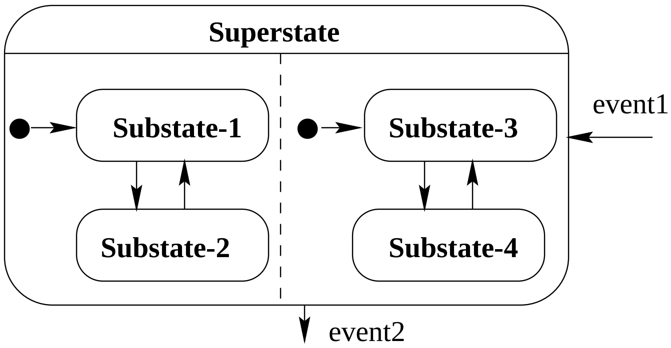
Ouput Event on a Transition:



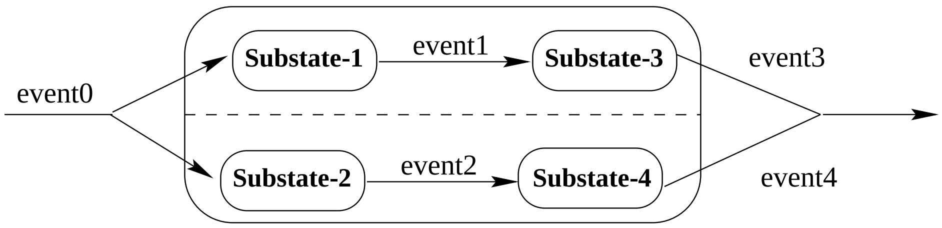
Sending an Event to Another Object:



Concurrent Subdiagrams:

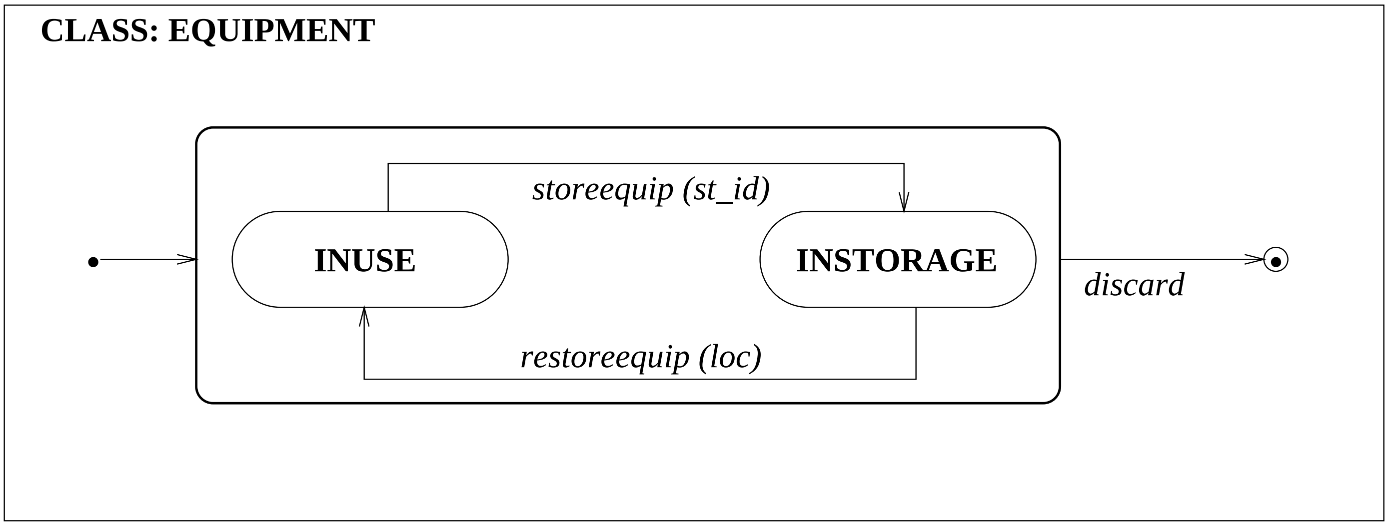


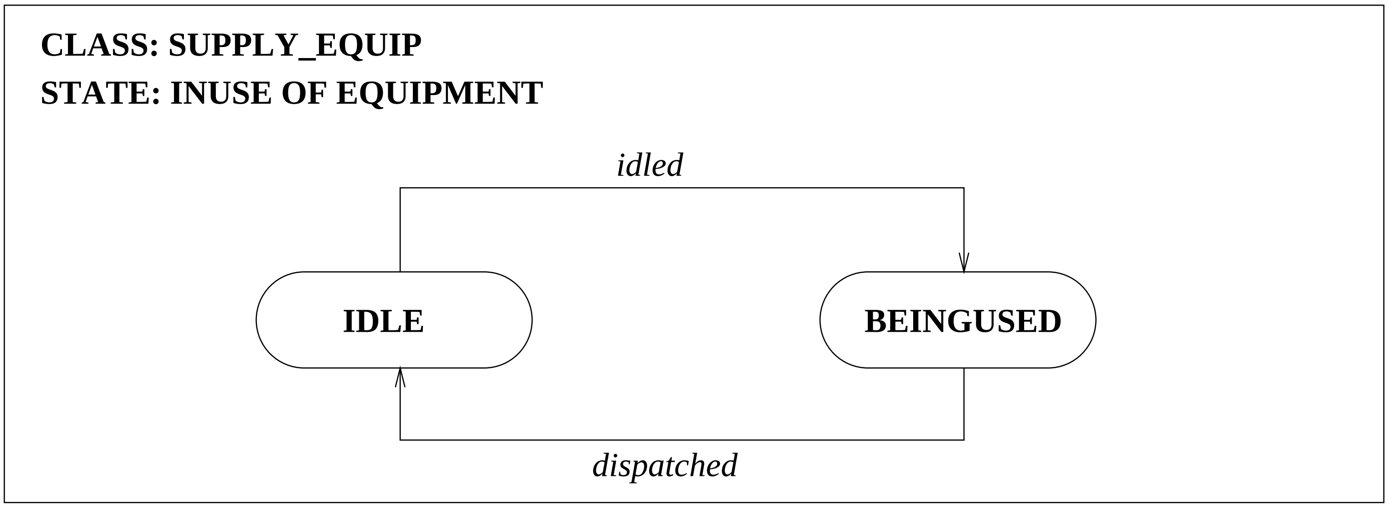
Splitting of Control: **Synchronization of Control:**

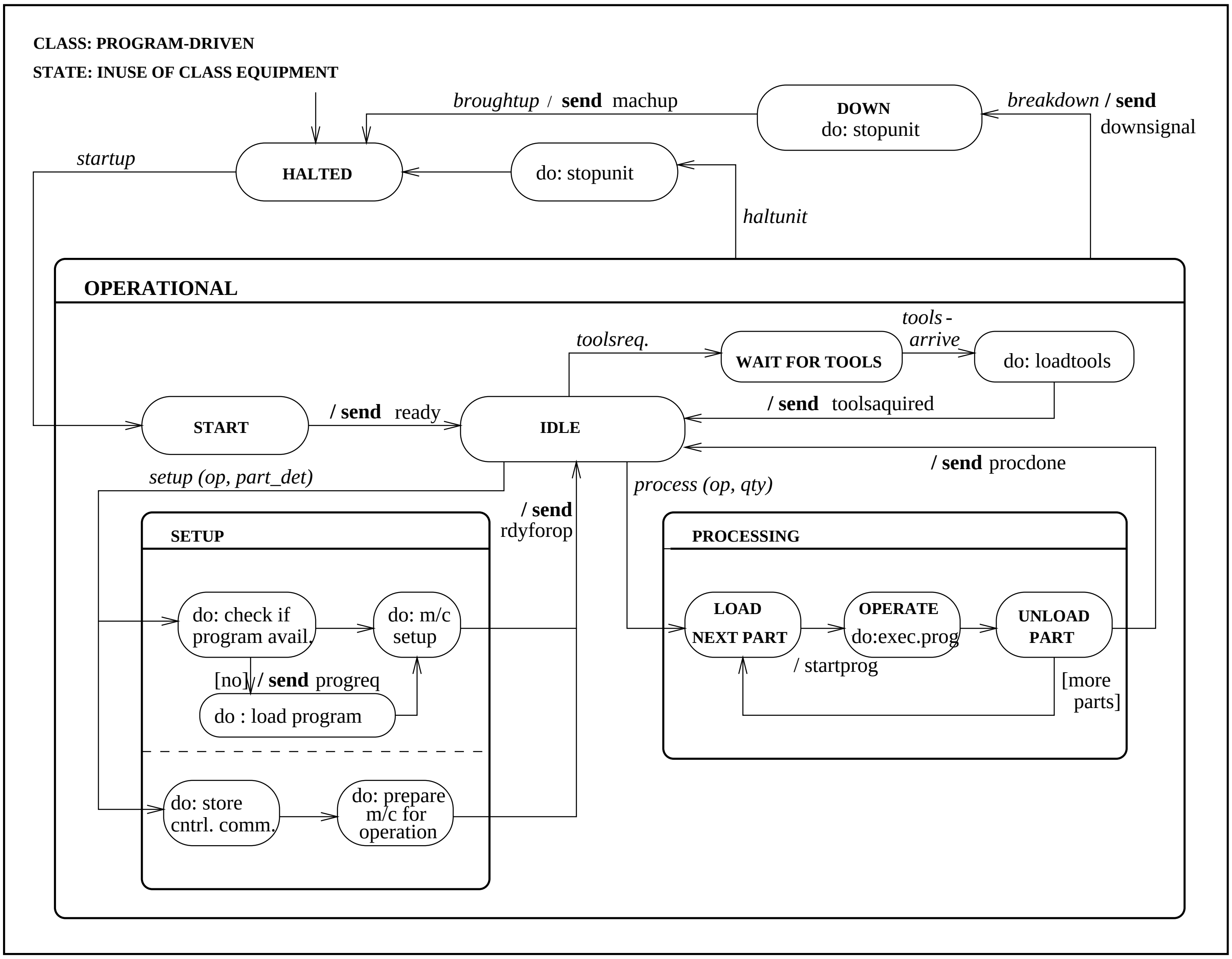


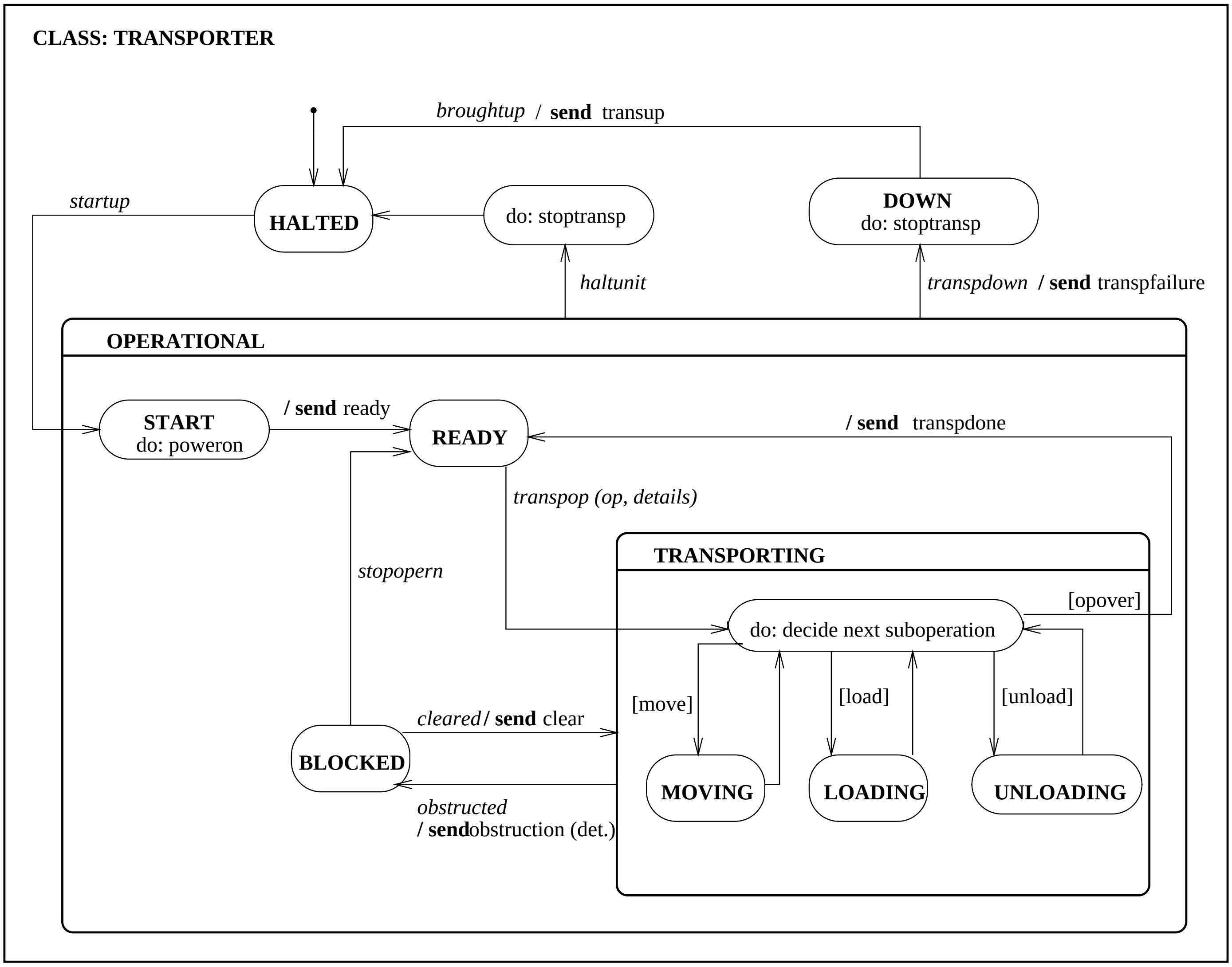
Appendix B

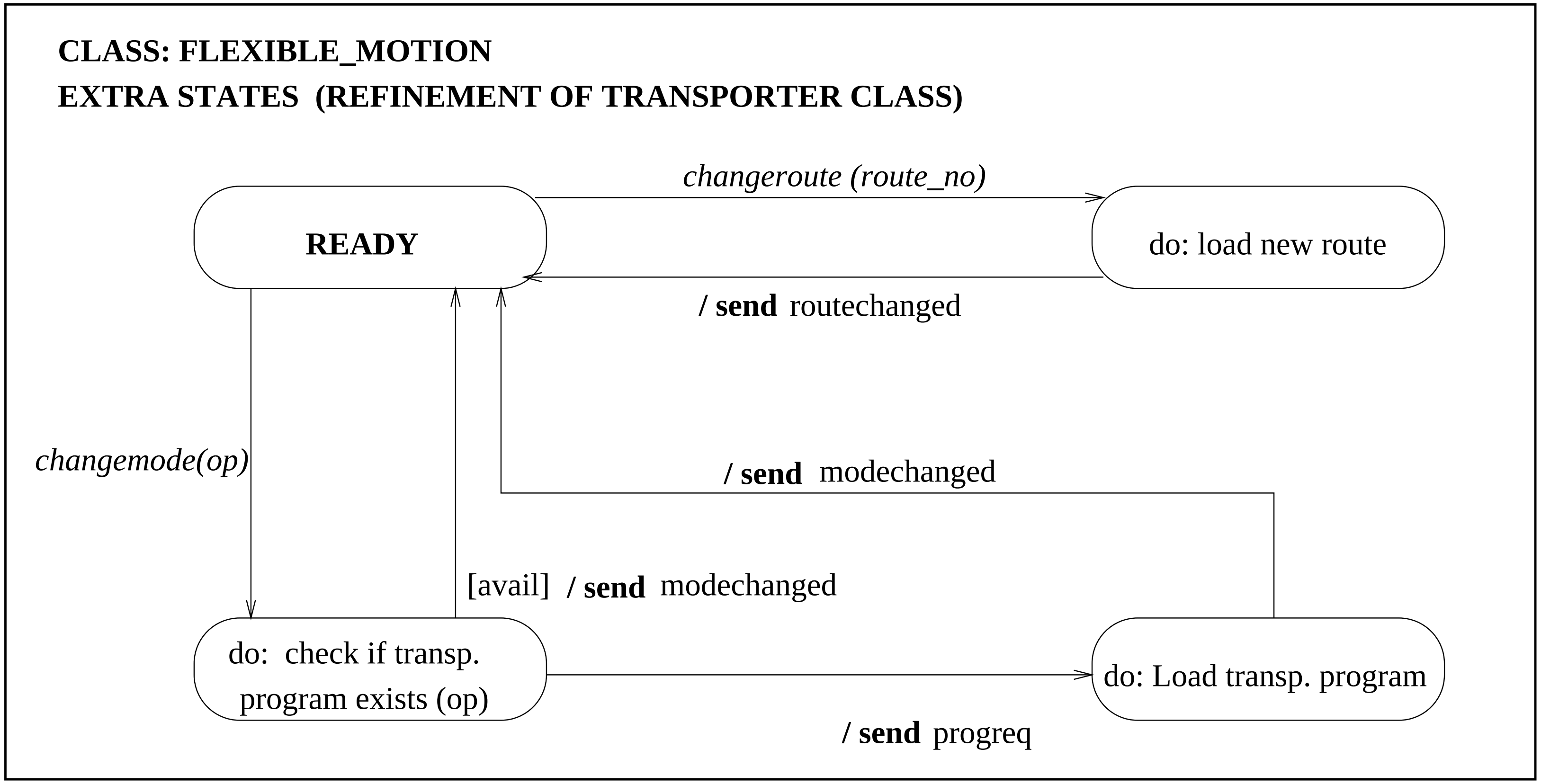
Dynamic Model for MIDAS

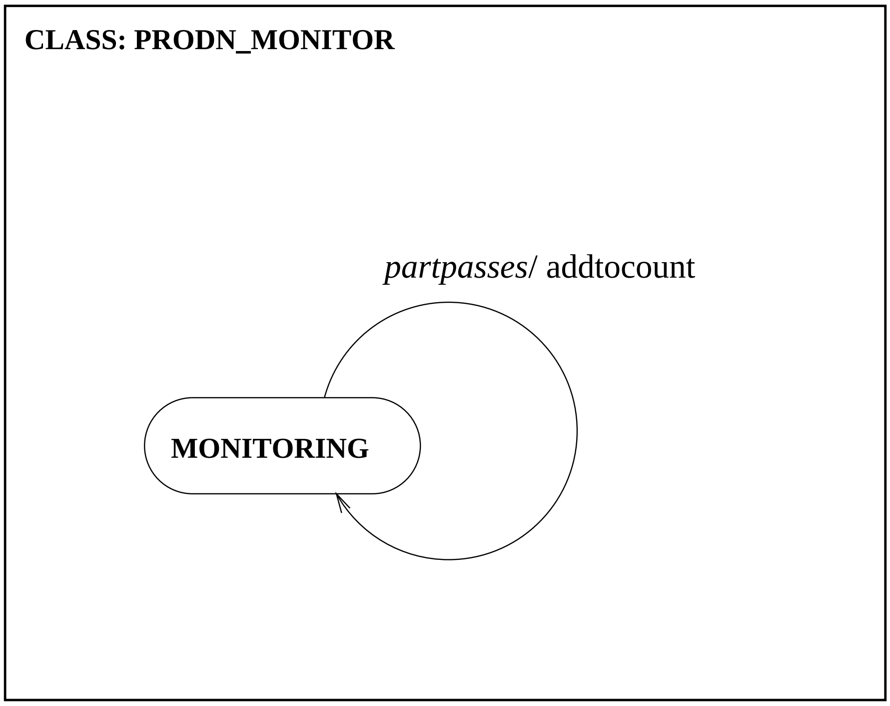

CLASS: EQUIPMENT
storeequip (st_id)
INUSE
INSTORAGE
discard
restoreequip (loc)

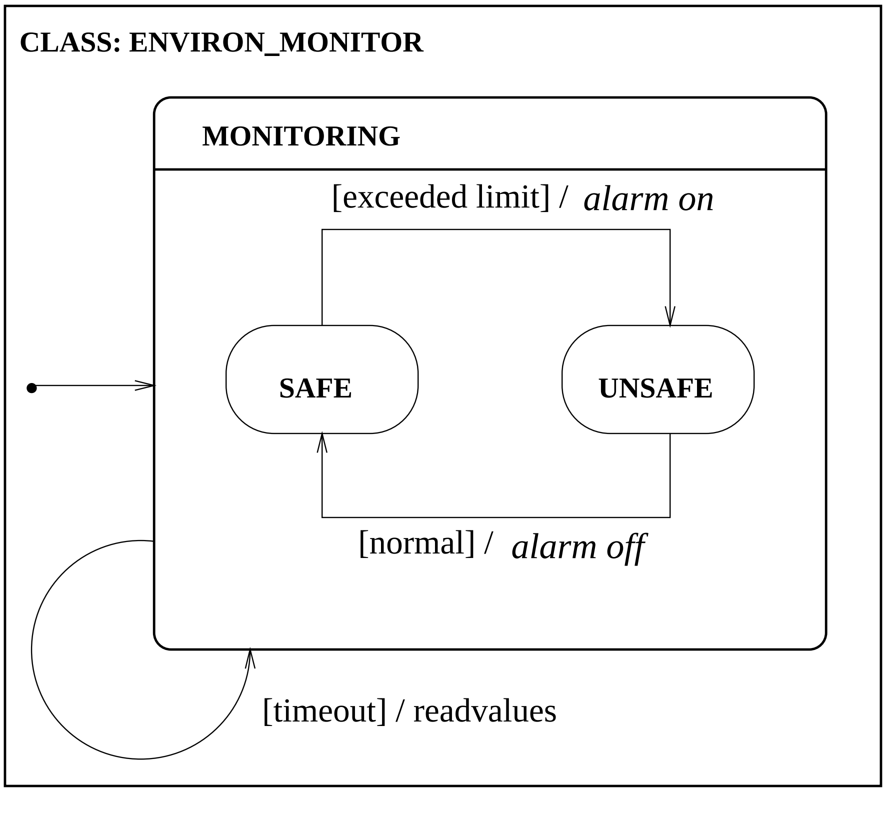

CLASS: SUPPLY_EQUIP
STATE: INUSE OF EQUIPMENT
idled
IDLE
BEINGUSED
dispatched

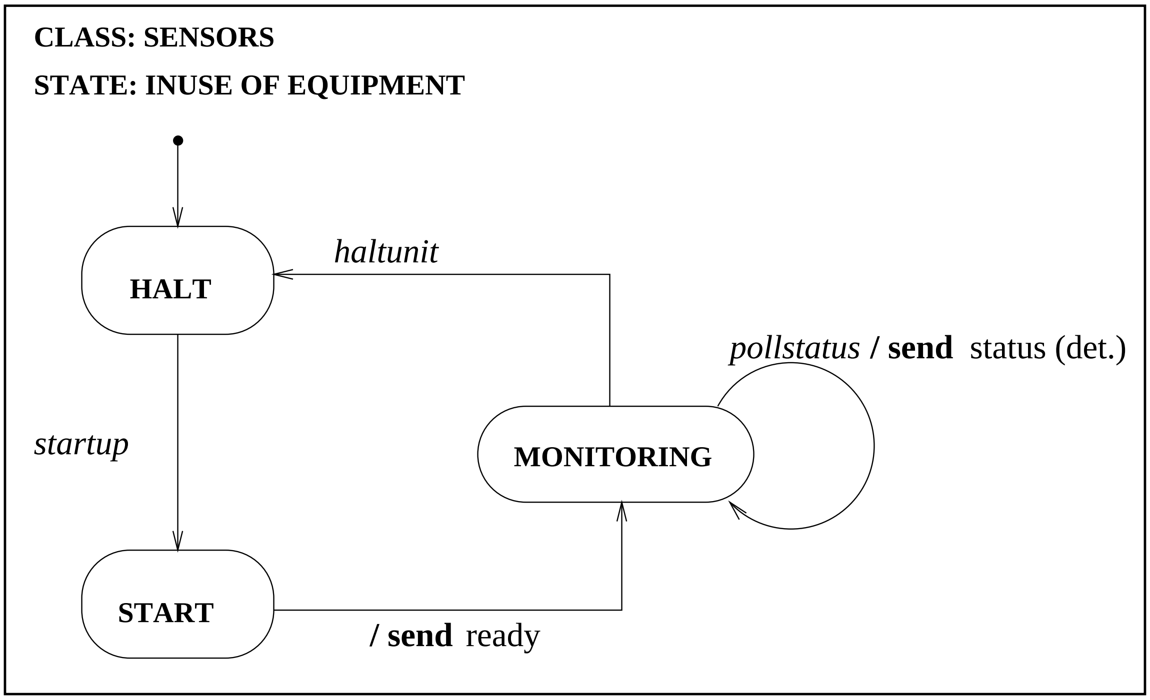

CLASS: PROGRAM-DRIVEN
STATE: INUSE OF CLASS EQUIPMENT
broughtup / send machup
DOWN
do: stopunit
breakdown / send downsignal
startup
HALTED
do: stopunit
haltunit
OPERATIONAL
toolsreq.
WAIT FOR TOOLS
tools-arrive
do: loadtools
START
/ send ready
IDLE
/ send toolsaquired
/ send procdone
setup (op, part_det)
process (op, qty)
/ send rdyforop
SETUP
do: check if program avail.
do: m/c setup
[no] / send progreq
do : load program
do: store cntrl. comm.
do: prepare m/c for operation
PROCESSING
LOAD NEXT PART
OPERATE do:exec.prog
UNLOAD PART
/ startprog
[more parts]

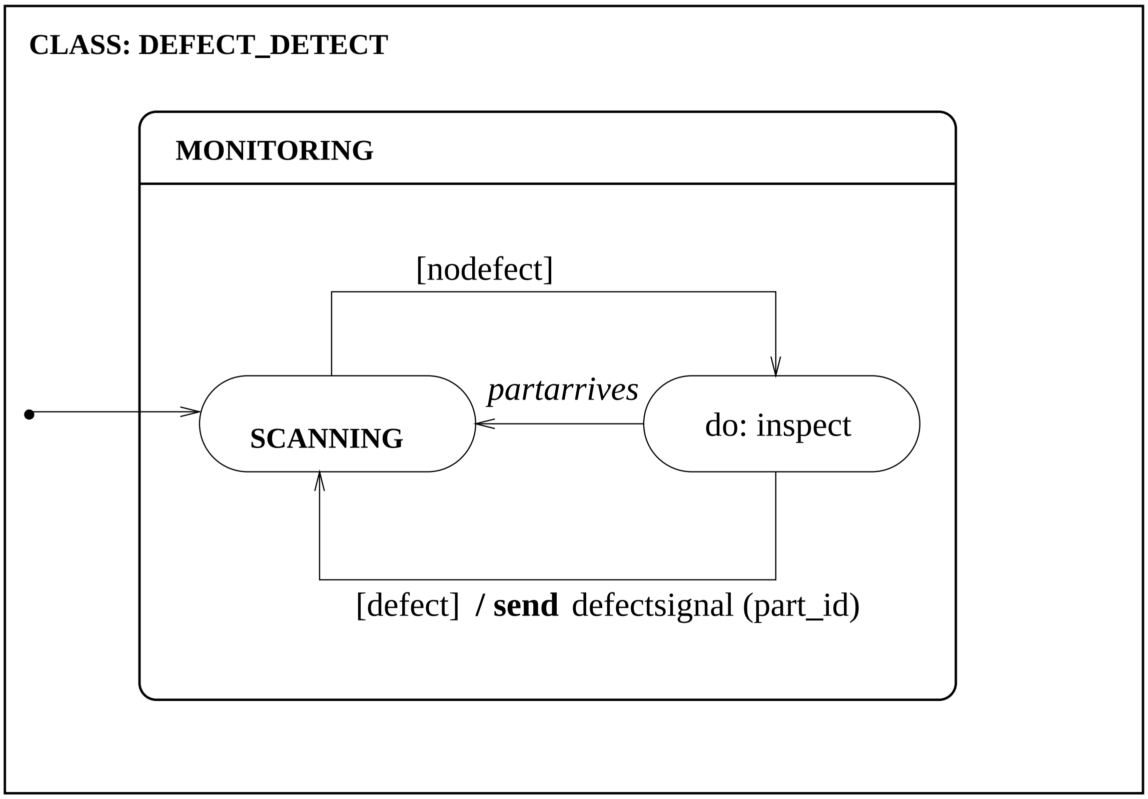

CLASS: TRANSPORTER
broughtup / send transup
startup
HALTED
do: stoptransp
DOWN
do: stoptransp
haltunit
transpdown / send transpfailure
OPERATIONAL
START
do: poweron
/ send ready
READY
/ send transpdone
transpop (op, details)
TRANSPORTING
stopopern
[opover]
do: decide next suboperation
[move]
[load]
[unload]
cleared / send clear
BLOCKED
MOVING
LOADING
UNLOADING
obstructed
/ send obstruction (det.)

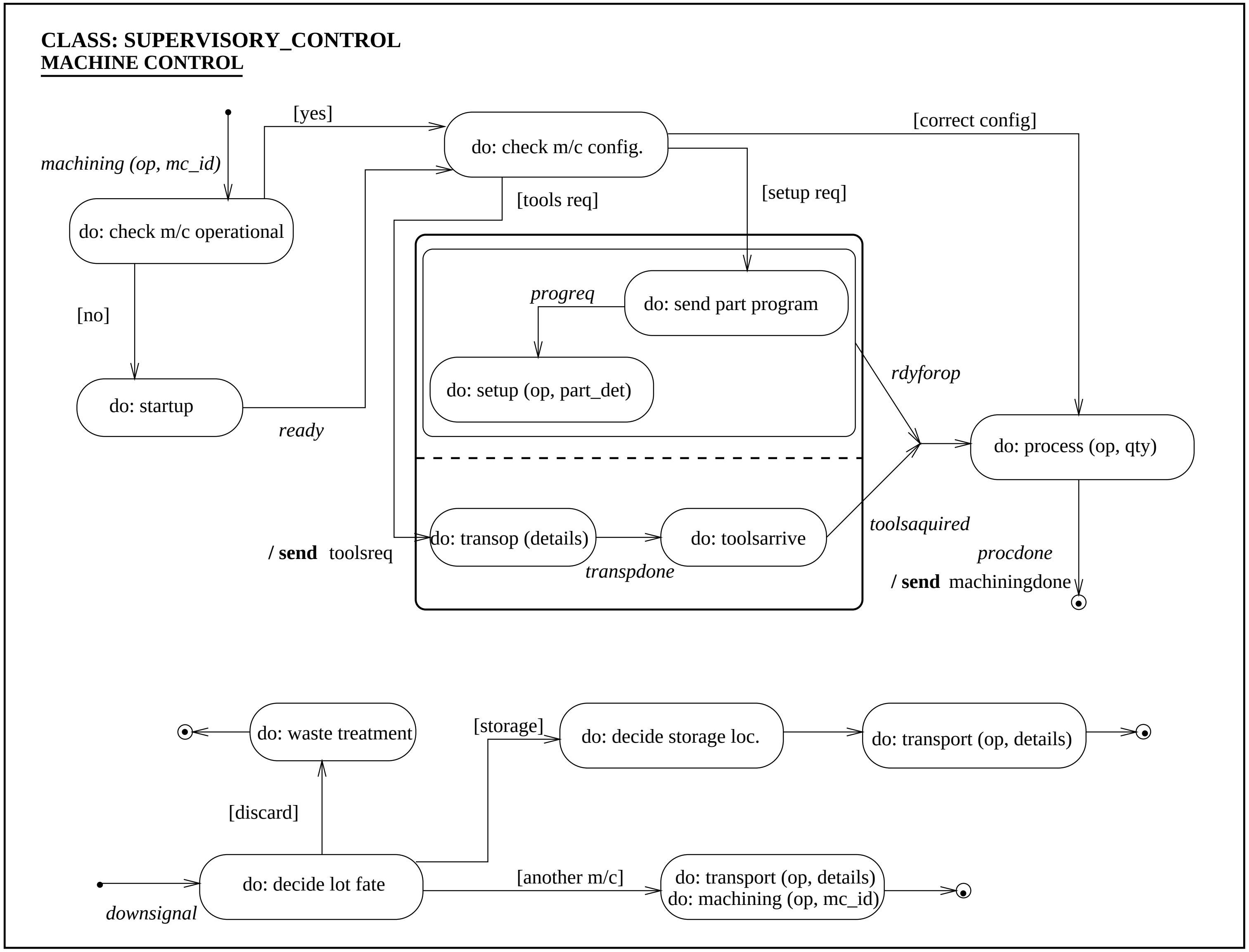

CLASS: FLEXIBLE_MOTION
EXTRA STATES (REFINEMENT OF TRANSPORTER CLASS)
READY
changeroute (route_no)
do: load new route
/ send routechanged
changemode(op)
/ send modechanged
[avail] / send modechanged
do: check if transp. program exists (op)
do: Load transp. program
/ send progreq

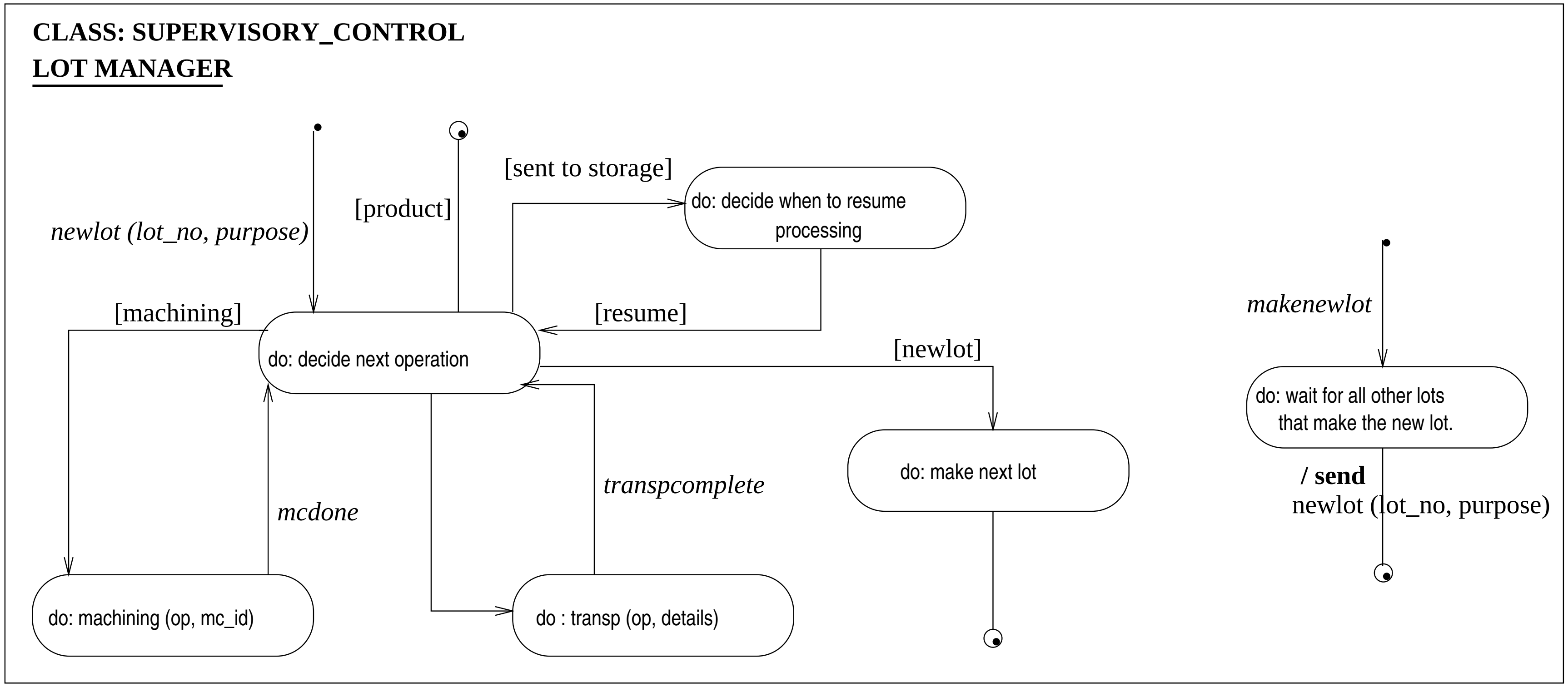

CLASS: PRODN_MONITOR
partpasses/ addtocount
MONITORING

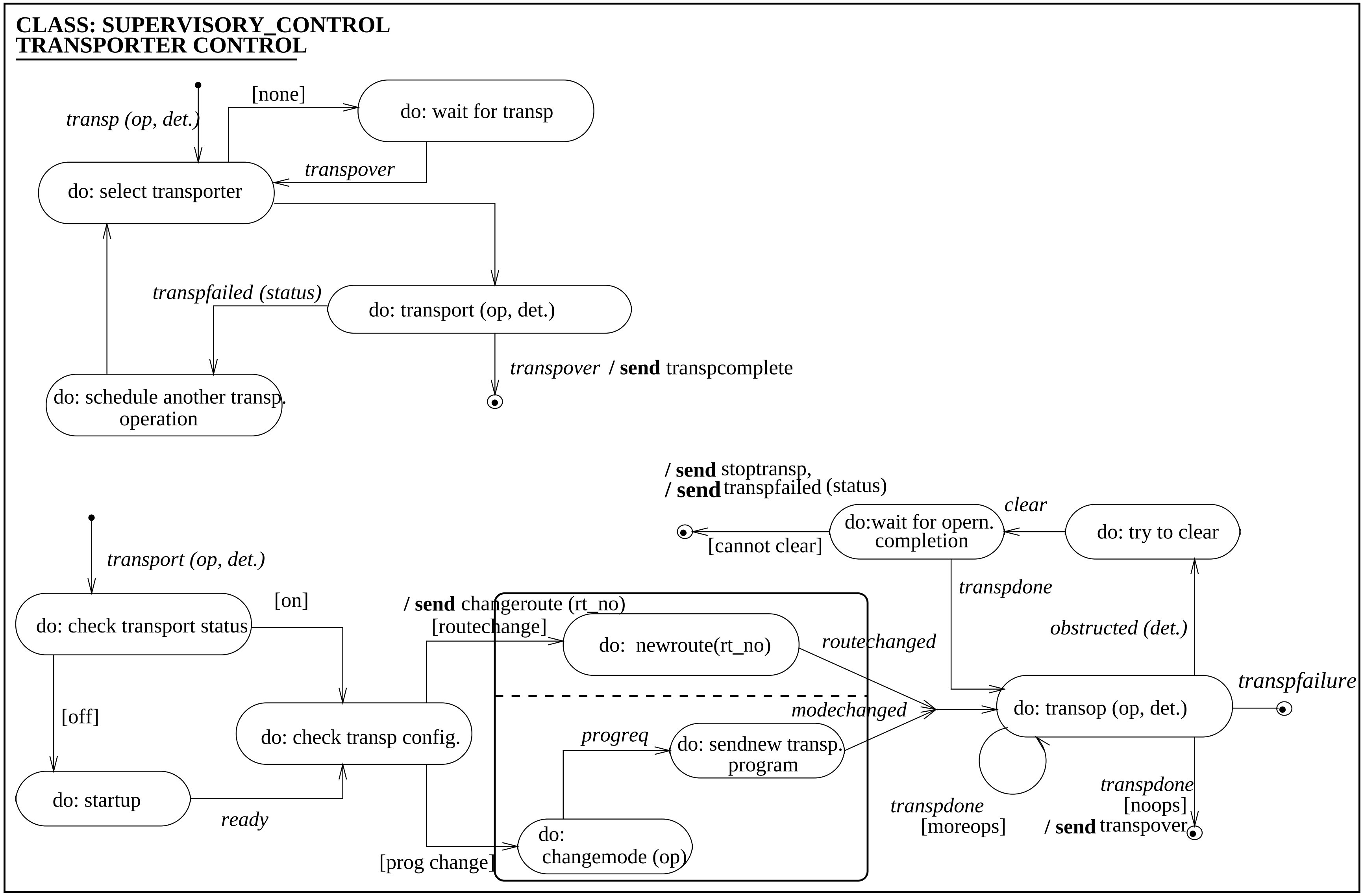

CLASS: ENVIRON_MONITOR
MONITORING
[exceeded limit] / alarm on
SAFE
UNSAFE
[normal] / alarm off
[timeout] / readvalues

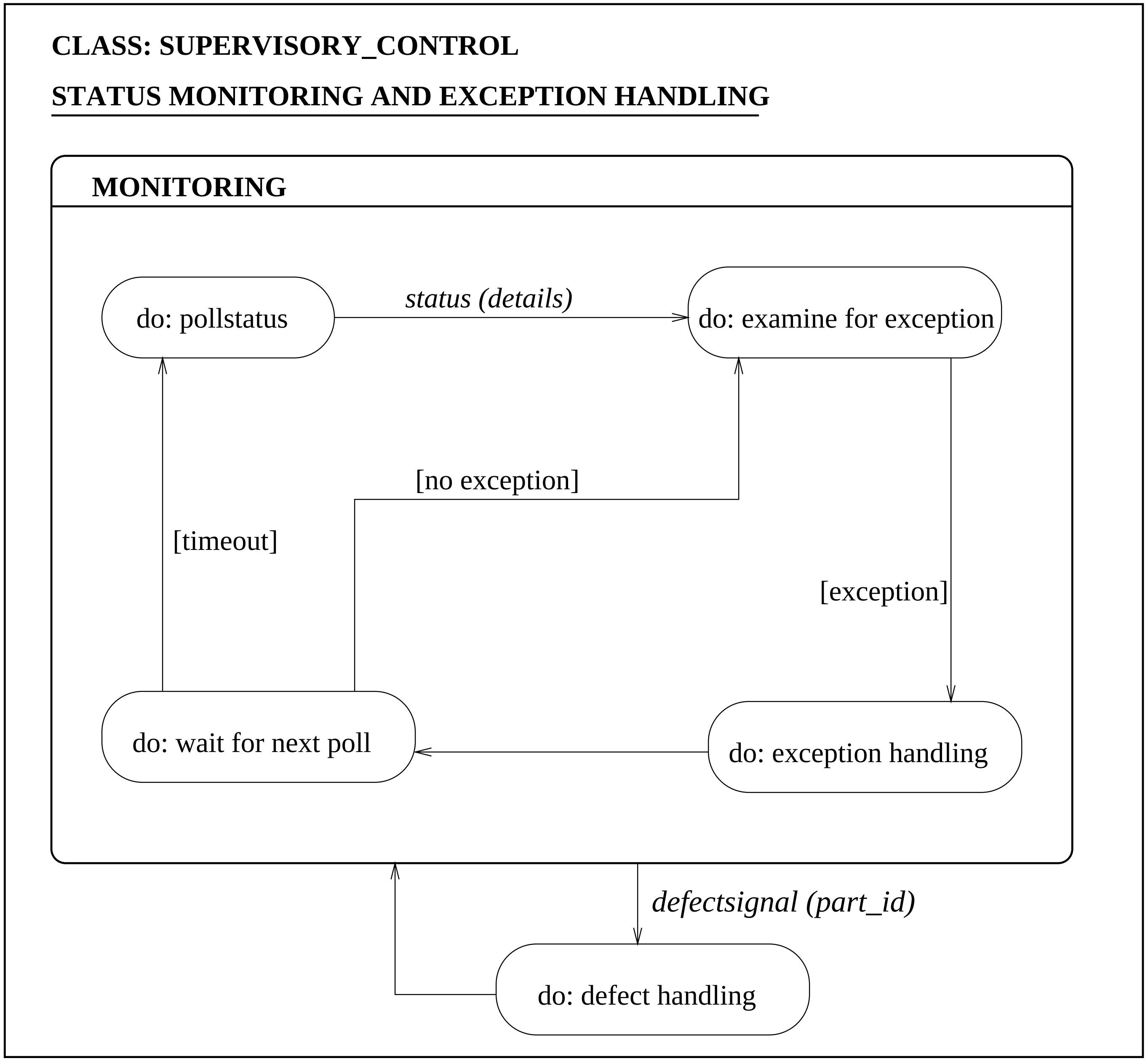

CLASS: SENSORS
STATE: INUSE OF EQUIPMENT
HALT
haltunit
pollstatus / send status (det.)
startup
MONITORING
START
/ send ready


CLASS: DEFECT_DETECT
MONITORING
[nodefect]
partarrives
SCANNING
do: inspect
[defect] / send defectsignal (part_id)


CLASS: SUPERVISORY_CONTROL
MACHINE CONTROL
machining (op, mc_id)
[yes]
do: check m/c config.
[correct config]
[tools req]
[setup req]
do: check m/c operational
[no]
progreq
do: send part program
do: setup (op, part_det)
rdyforop
do: startup
ready
do: process (op, qty)
/ send toolsreq
do: transop (details)
do: toolsarrive
transpdone
toolsaquired
procdone
/ send machiningdone
do: waste treatment
[storage]
do: decide storage loc.
do: transport (op, details)
[discard]
do: decide lot fate
[another m/c]
do: transport (op, details)
do: machining (op, mc_id)
downsignal


CLASS: SUPERVISORY_CONTROL
LOT MANAGER
newlot (lot_no, purpose)
[product]
[sent to storage]
do: decide when to resume processing
[machining]
[resume]
do: decide next operation
[newlot]
makenewlot
do: wait for all other lots that make the new lot.
mcdone
transpcomplete
do: make next lot
/ send newlot (lot_no, purpose)
do: machining (op, mc_id)
do : transp (op, details)


CLASS: SUPERVISORY_CONTROL
TRANSPORTER CONTROL
transp (op, det.)
[none]
do: wait for transp
transpover
do: select transporter
transpfailed (status)
do: transport (op, det.)
transpover / send transpcomplete
do: schedule another transp. operation
/ send stoptransp,
/ send transpfailed (status)
clear
do:wait for opern. completion
do: try to clear
[cannot clear]
transport (op, det.)
transpdone
[on]
/ send changeroute (rt_no)
[routechange]
do: check transport status
do: newroute(rt_no)
routechanged
obstructed (det.)
transpfailure
[off]
do: check transp config.
modechanged
do: transop (op, det.)
progreq
do: sendnew transp. program
do: startup
ready
transpdone
[moreops]
transpdone
[noops]
/ send transpover
do: changemode (op)
[prog change]


CLASS: SUPERVISORY_CONTROL
STATUS MONITORING AND EXCEPTION HANDLING
MONITORING
do: pollstatus
status (details)
do: examine for exception
[no exception]
[timeout]
[exception]
do: wait for next poll
do: exception handling
defectsignal (part_id)
do: defect handling