Design and Implementation of MIDAS - A Manufacturing Information 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

R. Bhuvaneshwari

Department of Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012

June 1996

Contents

Acknowledgements **iii**

Abstract **iv**

1 Introduction **1**
1.1 Motivation 1
1.2 The MIDAS System 4
1.3 Related work 6
1.4 Organisation of the report 8

2 The Manufacturing Domain **9**
2.1 Computer Integrated Manufacturing 9
2.2 The FMS 10
2.2.1 Organisational structure of an FMS 10
2.2.2 Control hierarchy 11
2.2.3 Database Management System for an FMS 13
2.2.4 Controller communications 13
2.2.5 A typical FMS scenario 14
2.3 Manufacturing data 15
2.4 Information retrieval 18

3 The Object Oriented Modeling **20**
3.1 The object oriented model 21
3.2 Building the MIDAS System 23

4 Analysis of the Problem **25**
4.1 Statement of the problem 25
4.1.1 Nature of domain data 26
4.1.2 Transactions for the manufacturing domain 27
4.1.3 Desired features of the MIDAS System 28

5 Designing the Data Model **30**
5.1 Identifying objects and classes 30
5.2 Identifying aggregations 32
5.3 Identifying other associations 34

5.4 Identifying attributes of objects and links . . . 36
5.5 Organising object classes using inheritance . . . 38
5.6 Verify access paths for likely queries . . . 40

6 MIDAS Sytem Design and Architecture 43
6.1 Need for new database design . . . 44
6.2 Features required of the backend DBMS . . . 45
6.3 MIDAS architecture . . . 46
6.3.1 Illustra backend engine . . . 48
6.3.2 MIDAS Server: . . . 51
6.3.3 MIDAS Server Interface: . . . 52

7 Object Design and Implementation 55
7.1 Design optimisations . . . 55
7.2 Design of associations . . . 57
7.2.1 One-Way associations . . . 57
7.2.2 Two-way associations . . . 57
7.2.3 Link attributes . . . 59
7.2.4 Ternary associations . . . 59
7.3 Design of aggregations . . . 60
7.4 Representation of object attributes . . . 60
7.5 Implementation of MIDAS . . . 61

8 A Manufacturing Plant Example 64
8.1 Description of the Coffee Making Plant . . . 64
8.2 Choice of the example . . . 65
8.3 Implementation of the Coffee Plant model . . . 66
8.3.1 Object classes used . . . 66
8.3.2 Creation of a knowledge base . . . 68
8.3.3 Tracing event sequences . . . 69
8.3.4 Interface . . . 69
8.3.5 Implementation details . . . 72
8.4 Adapting MIDAS . . . 73

9 Conclusion 76
9.1 Summary . . . 76
9.2 Suggested extensions . . . 77

Bibliography 78

A Rumbaugh Notation 83

B Coffee Plant Display 85

Acknowledgements

I take this oppurtunity to thank my project guide, Dr. Jayant Haritsa for his excellent guidance throughout my project. I am grateful to him for his advice and suggestions at every stage. I thank him for the DSL lab which provided us an excellent environment to work in. I also thank Dr. Y. Narahari for his extremely useful suggestions.

I wish to thank my labmates Arun, Ramesh and Binto for their help during the submission of my project. The numerous discussions I had with Krishna were very useful.

I thank all my batchmates for the nice time we had together in IISc. A special thanks to Hina, Suthi and Renu for making my stay pleasant and memorable.

Finally, I thank my parents, my brothers and my grandparents for their encouragement and support from ever since I can remember.

Abstract

Automated factories have grown from a concept to a viable undertaking and are gaining popularity by the day. Computer Integrated Manufacturing (CIM) integrates all phases of manufacturing from product conception and planning to shipping and delivery. One of the major subsystems for CIM is the Flexible Manufacturing System (FMS) for production control. The FMS is an automated system based on group technology manufacturing which has a central computer control and automated material handling designed to cater to variability in product mix, volume requirements, and design changes. The magnitude and complexity of the data generated in an automated system requires efficient data management. This in turn requires a suitable data model for representing manufacturing data. The manufacturing industry has been using the relational system which is inadequate for managing this data. Recent research has focussed on the object oriented approach for modeling and programming which is well suited to the requirements of the manufacturing data. The Manufacturing Information DAtabase System (MIDAS) project was developed to address this issue of data management in the manufacturing domain.

In this report, we describe the design and development of MIDAS, a data management and control system customised for flexible manufacturing systems. The MIDAS database supports complex manufacturing objects. The database has been implemented on Illustra, an Object Oriented Relational DBMS. Currently, MIDAS comprises a backend DBMS for data handling, a simple scheduler, a view generator for displaying the plant status and other monitoring information, and a simplified controls panel for operator intervention. It also includes routines to access the database and control routines to handle simple events. With the concepts of Total Quality Control(TQC) and zero defect production catching on, it is necessary to have complete control of processes at every stage of manufacturing. Hence, MIDAS provides comprehensive views of the plant, processes and workpieces which can be utilised for analysis and control of the plant.

MIDAS has been tested through the example of a simulated coffee manufacturing plant. MIDAS is the controller of this automated plant and manages event handling as well as data handling. MIDAS has a graphical user interface, built using X and Motif, for interacting with the automated plant.

Chapter 1

Introduction

1.1 Motivation

Complete shop-floor automation, that is, a “lights-out” plant was once a utopian dream. The extensive use of computers and intelligent systems for the different functions in a manufacturing enterprise has helped the manufacturing world come close to making it a reality. The bottom up evolution of industrial automation has resulted in *islands of automation*, that is, the different subsystems are isolated automated units [Sepe87]. The concept of Computer Integrated Manufacturing (CIM) aims to bridge the gap between these subsystems, through coordinated information flow between them. This concept denotes the pervasive use of computers to design products, plan the production, control the operations and perform the various business related functions needed in a manufacturing firm [Groo87]. The Flexible Manufacturing System (FMS) is one of the subsystems of CIM, which caters to the actual manufacturing of the product. An FMS is an automated system suited to versatile manufacturing of products of the same family. It is designed to cater to manufacturing trends such as high variability in product mix, in volume requirements, and in design changes. This is done by using versatile machines that are capable of a number of operations and by increasing the capacity of production.

There is a variety of data that has to be dealt with in the context of automated manufacturing. The data in a manufacturing database may be classified as shown below.

- Structural data

  This data comprises the "static" or the slowly changing data. This category of data provides the basic framework of different entities that describe the plant. It includes

  - Configuration data such as plant layout details, cell configurations, and warehouse configurations
  - Equipment information such as details of the machines, tools, and accessories such as pallets, containers, and fixtures
  - Operational data such as part programs, production schedules, maintenance schedules, route sheets, and software for different applications
  - Product data such as list of company products, part designs, process plans, and bill of materials
  - Production data such as lots and their description
  - Inventory information such as stock levels of raw materials, semi-finished parts, subassemblies, fuel, and coolants
  - Information describing customer orders, customers, purchase orders, vendors, and account receivables and payables

- Status Data or Temporal Data

  This is the information received from the plant monitors and sensors. This describes the "health" of the plant in terms of the plant's usage and operational quality. Machine status, monitor or sensor values, job status, position of a material handler, lot status, inventory levels, etc. fall into this category. Status data have limited time utility with respect to the immediate operation of the plant, but a *history* of the data values may be maintained. Performance and fault analysis of the manufacturing processes are based on this category of data. Status data varies quickly as compared to structural data and is stored along with the time at which the data was monitored. Thus, it has a time stamp associated with it. The status data forms the bulk of the data generated by the automated shop floor. Due to the high arrival rate of the status data, it requires increasingly large storage space. Hence, periodically the old status data is archived or a version of coarser granularity is stored.

- Performance data

  Information derived from the status data are utilised for analysis of the plant's operations. Various performance parameters such as throughput, waiting times, transportation times, machine utilisations, and workshop loads form the basis of evaluation of the plant.

- Control data

  The Control data captures the current settings of the machines and plant controller. In addition to the current parameter settings, the control database also stores "profiles" that reflect appropriate settings for a variety of common manufacturing patterns and plant configurations.

The volume of data generated in an automated plant is of the order of gigabytes per day. For example, if there were 1000 sources generating sensor information at the rate of 10 bytes per second, the amount of data in a day is about 1.2 MB. The magnitude and variety of data encountered in the manufacturing domain call for efficient data management for information retrieval. The Manufacturing Information DAtabase System (MIDAS) project aims to develop a database system design tuned for supporting flexible manufacturing systems. The goal of the MIDAS System is to be able to handle both data management and control of a flexible manufacturing system. In this thesis, we describe the first phase of development of the system. We have identified and modeled the various entities of the manufacturing domain and designed the MIDAS architecture. The MIDAS System was built using the Object Methodology Technique (OMT) [Rumb+91]. The data model was implemented and the system was finally tested for a simple automated environment.

1.2 The MIDAS System

In this section, we briefly describe the development of the MIDAS System. The two important aspects of an automated system are shop floor control and data management. MIDAS uses a DBMS as a backend engine for managing the data. It makes use of the features of the backend engine to provide embedded control. In order to enable MIDAS to completely take over the factory control, additional control routines and expert systems have to be integrated into the system. Currently, MIDAS comprises a backend DBMS for data handling, a simple scheduler, a view generator for displaying the plant status and other monitoring information, and a simplified controls panel for operator intervention. It also includes routines to access the database and control routines to handle simple events.

The principle of the MIDAS system is to make the database faithfully follow all the events in the automated shop floor. The data generated by events in the factory floor are communicated to the MIDAS System, which records the data along with the time at which the data was generated. The data may cause MIDAS to trigger actions that initiate appropriate activities in the factory. This data generated by the factory events is essentially the status data and is used to display information to the operator supervising the plant. In the first stage of developing the MIDAS System, an Object Model of the manufacturing domain was constructed to establish the relationships between the various entities in the domain. Subsequently, the architecture of the system was designed.

The MIDAS architecture comprises three components, namely, the *Illustra Data Server*, the *MIDAS Server*, and the *MIDAS Server Interface*. This can be seen in Figure 1.1.

- *Illustra Data Server*

  Most legacy applications for manufacturing use the traditional relational data model. The newly developed system must be able to easily interface with the existing applications. Hence, MIDAS was built over Illustra, a Relational DBMS with Object Oriented features. Illustra is a commercial object relational database with support for trigger mechanisms. The Illustra backend engine forms the lowest layer of MIDAS. It offers two interfaces, the *Interactive Query Interface* and the *Application*

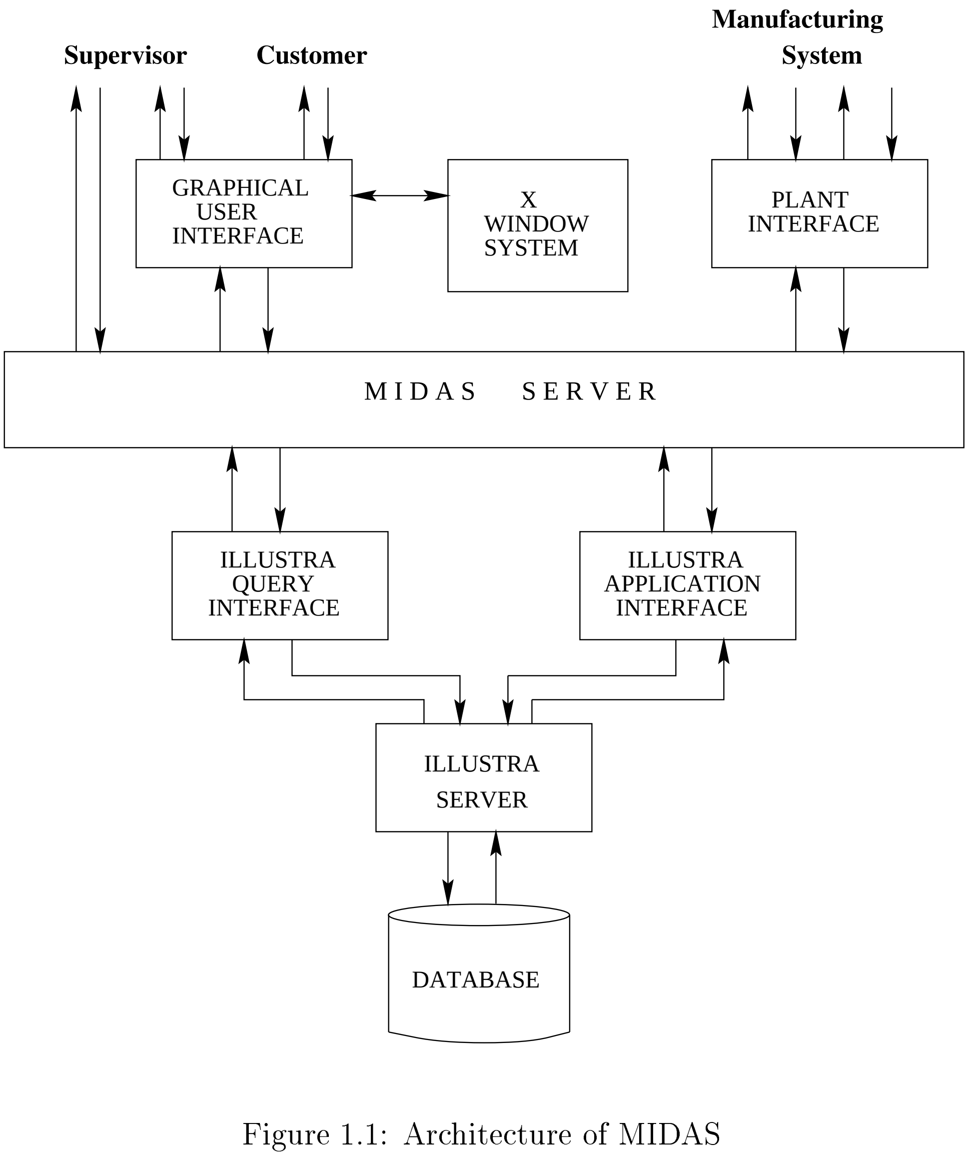


Figure 1.1: Architecture of MIDAS

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 Server*

  The MIDAS 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 all the control routines for plant control as well as routines to access the database. The control routines use the Application Programming Interface of the Illustra data server 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, Motif]. 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.

As the first step towards implementation, MIDAS has been tested through the example of a simulated coffee manufacturing plant. MIDAS handles data management, view generation as well as the control aspect of this plant.

1.3 Related work

The design of the MIDAS system involves many issues such as selecting a suitable model for the domain data, implementing the model, making decisions on the type of information to be stored and retrieved, and providing suitable interfaces to the system.

There are several books describing the features of an FMS. A comprehensive overview of an automated manufacturing system is provided in [VN92]. It also discusses the different types of flexibility that an FMS can have. An introduction to the Computer Automated Manufacturing (CAM) is presented in [John87]. In [Lugg91], the author discusses flexible manufacturing cells and systems, and FMS principles and practices. Principles of CIM and related issues have been discussed in [Groo87].

Most manufacturing systems use the relational model for data representation. The object oriented approach of data modeling gives a more natural method of representing entities of a domain. The books [Booc94, Rumb+91] describe object oriented methods of data modeling. 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 generalise a product family [KR95]. ROSE, a data 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 engineering objects [HS89]. Both the systems referred to above use a relational DBMS for data storage and management.

In a manufacturing domain, the temporal data may be that of the processes or of the lots of the product. A method to combine these data to present a comprehensive view to the operator is described in [Taka+95, Taka+96]. The authors describe the *Plant, Process*, and the *Lot* view which are of benefit to the operator monitoring the process. MIDAS provides views on similar lines.

A visit to Widia, Special Purpose Machines Division, gave a valuable insight into automated manufacturing. This division of Widia, uses an FMS for manufacturing a variety of tools. The article [Sptr93] is a collection of four case studies on flexible factories which describe flexible manufacturing facilities in industry. It helped to understand the nature of flexibility required in an FMS.

The intention of MIDAS is not to provide an entirely new set of control routines to control the plant, but to incorporate existing ones so that, the MIDAS Server now capable of control as well as data management. The control programs used in an automated plant are specific to the plant. Hence, we have not considered issues related to actual control equipment and plant control. However, we provide methods to invoke these control routines from MIDAS.

1.4 Organisation of the report

The rest of the report is organised as follows. Chapter 2 provides a brief introduction to the FMS. Chapter 3 discusses the merits of object modeling and describes the stages of development of the MIDAS system. After a careful analysis of the application domain, we arrive at the problem statement in Chapter 4. Details of the object model are presented in Chapter 5. Chapter 6 deals with the design and architecture of MIDAS. We discuss the object design phase and the implementation of the model in Chapter 7. Chapter 8 describes the implementation of MIDAS for a simulated coffee plant. Finally, we conclude and outline future research avenues.

Chapter 2

The Manufacturing Domain

In this chapter, we provide an introduction to the various aspects of a flexible manufacturing system. Firstly, we describe the organisational and control structure of an FMS. Subsequently we describe the issues regarding the manufacturing data and retrieval of manufacturing information.

2.1 Computer Integrated Manufacturing

Computer Integrated Manufacturing (CIM) is a concept which calls for coordinated participation of computers in all phases of a manufacturing enterprise from design of product, production planning, and production control to business activities like marketing, order entry, and customer billing [Groo87]. Achieving CIM implies integrating all phases of the business process.

Some manufacturing subsystems used in the various stages of manufacturing are mentioned in this section [John87, VN92, Sepe87]. The Computer Aided Design (CAD) system uses computers and graphics for automating design of product and tools. 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 The FMS

The Flexible Manufacturing System can be defined as an automated system based on group technology manufacturing having a group of automated machines, material handlers and a central computer control to dynamically balance resource utilization so that the system can adapt automatically to changes in parts production, mixes, and levels of output [Lugg91]. Group technology is a manufacturing philosophy in which similar parts are identified and grouped to take advantage of their similarities in manufacturing and design [Groo87]. The FMS drives the total system and provides the decision criteria to anticipate resource needs, schedule production and respond to both planned and unanticipated real-time activities.

In the following subsections, we present discussion on the FMS. The main reference for this section was from [VN92].

2.2.1 Organisational structure of an FMS

The FMS, which is a structured manufacturing system consists of a set of manufacturing cells which are dedicated to performing a set of related operations for a family of parts. Intercell movement of materials is handled by material handling system (MHS) which governs the movement of the transporters such as Automated Guided Vehicles (AGV) and conveyors. Each cell comprises of one or more automated machines and material handlers which cater to part movement within the cell. The time for part movement within a cell is negligible or very small compared to the total lead time or the time for production of the product. The activities within a manufacturing cell are controlled by the cell controller.

2.2.2 Control hierarchy

The control system is typically an interconnected network of the programmable controllers, cell controllers and the supervisory computers. [VN92]. The software resident in the various controllers has the capability to enable automated operation as well as monitoring and diagonistics.

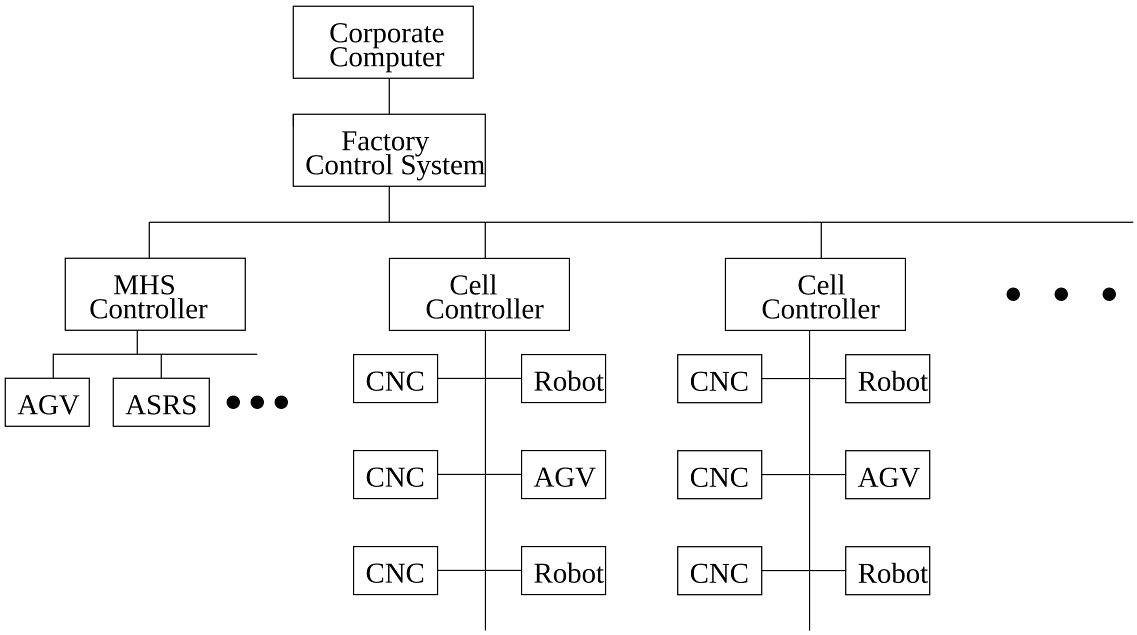


Figure 2.1: Control hierarchy for an FMS

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 is the highest in the hierarchy. It coordinates the different manufacturing activities. It manages the long term planning activities such as master production schedules. Under this hierarchy 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 cell assignment of workpieces and movement of material handlers between the cells. It guides the workpieces from the loading station through the cells to the unloading station. It has direct access to the DBMS consisting of part programs, tool databases, routings, and equipment status information. The system has also incorporated into it expert systems to handle activities such as short term scheduling, part routing, deadlock resolution for the transporters and decision support to handle unanticipated events.

The Cell Level Controller handles the control of machines and the routing of workpieces within the cell. It has to make decisions regarding which part is to be processed next, machine loading and job sequencing for service by the MHS or for processing by the machine. It also has the capacity to detect failures in equipment in the cell and take corrective action. Error conditions are signalled by intelligent resources such as expert system or by sensors. In the situation of a distributed database with each cell having its own database, the cell controller is responsible for retrieving part programs, routing for the part types, tool databases, etc. Status information of the cell is also stored at this database.

The Equipment Control System handles monitoring and control of the individual equipment namely, the machines, AGVs, and robots. They translate the commands issued by the cell controller into specific instructions to the actuators of the equipment. The cell controller downloads part programs to equipment controllers and also collects status information from them.

Thus, there is a hierarchical control link from the corporate planning to the shop floor control and an information link flowing up.

2.2.3 Database Management System for an FMS

The CIM database comprises the CAD database which contains design data for the products, the MRP database which contains data required for planning production, CAM database which concerns with the status data of the plant and part programs, and the ACC database which stores the financial and administrative data. Within the FMS, data concerning the manufacturing aspects of the product are stored. The database may be centralised or distributed. In the centralised case, all the data is stored at the factory level. Most automated plants use a distributed environment [VN92, Sepe87]. The factory level database of the plant stores general information about the plant, the products, plant configuration details, schedules, software for expert systems, MHS, etc. The databases local to the cell store temporal information, the information such as part programs for machines in the cell and details of tools and other equipment within the cell.

The industry currently uses distributed, heterogeneous databases for storing the data. 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.

The relational model is not adequate to represent the manufacturing data. 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 uniform storage of the heterogeneous data of the manufacturing world.

2.2.4 Controller communications

A variety of messages are passed between the different controllers [VN92]. The part programs and operational data are stored in a centralised 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 such as part type being worked on, number scheduled, number waiting, and equipment status information 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 usage, fixtures, raw and semi-finished workpieces are stored in the database through the cell controllers or the FCS. Realtime data for time-critical operations such machine feedback, quality control data, monitoring and diagnosis information, AGV routing, and part flow data are also transferred to the supervising controller.

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 sheets from the database for the parts. The FCS decides the routing based on rules or 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. It directs the MHS to move the part to the appropriate cell. The cell controller decides which machine to load it on, and retrieves the part programs and tools description from the local database if one exists or requests the program from the FCS. The cell controller manages routing of the parts within the cell. It monitors the progress of parts in the cell and reports its performance to the FCS. It decides the actions to be taken when the values of the variables describing the machine's status are not available.

When the cell has finished processing a part, the FCS guides it 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 on a part are over, the part is defixtured and sent to the storage. The FCS also has to make decisions regarding the choice of the AGV to transport the part.

2.3 Manufacturing data

This section aims to highlight the following issues related to data in the manufacturing domain.

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

The data in the context of a manufacturing domain is heterogeneous in nature. As explained in the earlier chapter, this data may be classified into four categories the Structural or "static" data, Status Data or Temporal Data, Performance data, and Control data.

An important issue is the representation of these data. We can easily observe that the structural data constitute the *objects* of the manufacturing domain. The configuration data, equipment description, operational data, product data, inventory information, customer information, and purchase details constitute information which is relatively static. Each object has a set of attributes associated with it. The status data is basically the temporal value of these attributes or *states* of the object. The status data changes rapidly across time. The performance data is derived from the temporal data. The control data also qualifies a *object*. It comprises of a set of values which describe the *setting* attribute. Software programs, stores description, plant configuration are some objects which do not have any temporal feature. We describe below the nature of the data associated with some objects such as lots, processing units, material handlers, inventory, tools, and storage units. All except the last *object* have permanent as well as temporal features associated with them.

- Lot information

  The workpieces are handled in units called lots which may comprise one or more workpieces. Setup time and setup costs are incurred when the manufacturing system switches to a new product type. The setup time includes time for tool changing and

preparing the workplace. In order to minimise the average setup time, the parts are processed in lots or batches. It is desirable to reduce setup times. In an ideal FMS, lot size is 1 unit. The transfer of work-in-process parts from one machine to another is done in units of the *transfer* lot size. A lot is identified by a unique lot number. Associated with a lot are attributes such as description of the lot and the time taken for this lot at every processing stage.
Lot renaming: A manufacturing process may split a lot into a number of smaller ones or, the process may combine two or more lots to form a new lot. Tracking the lots and maintaining its history is yet another issue requiring attention. Depending on the processes in the domain, it may be necessary to rename the new lot and maintain links to the parent lot(s).

- Processing Units information

  Processing units in a manufacturing cell may be manual work centers, or an automated machine. The lots are processed at the processing units.

  Among the structural information stored regarding the processing unit are configuration details, the operations it can perform, time taken for each of the operations, lot size that it can process, 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* or *UNLOADING* or *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 analyse the general "health" of this unit.

- Tool information

  Tool management is another issue to be dealt with in a manufacturing environment. The processing units may share tools for performing operations. Hence tool scheduling is very important in an FMS. The tools are identified by a unique code number. Tool description, functionalities and usage are some of the structural data of the tools to be stored in the database. Temporal information of tools concerns their allocation to processing units.

- Material Handlers information

  The material handlers are the devices which cater to movement of parts from one location to another. They may be automated or manually driven and may have fixed or flexible routing. Like the processing units, the material handlers may be in any one of the following states, *IDLE*, *LOADING*, *UNLOADING*, *TRANSPORTING*, *DOWN*. Information stored concerning the material handlers are their identification number, nature of their movement and the route of movement. Performance parameters include utility and rate of transfer.

- Storage Units

  Temporary buffers at the processing units hold the lots temporarily till they are processed. Their capacity is limited. A full output buffer blocks the processing units. A full input buffer directs the controlling software to suspend further allocation of jobs to this processing unit.

  The records for long term storage units contain besides their identifying name, a capacity and specification of environmental conditions in the stores. Inventory costs contribute significantly to product cost and lowering inventories is desirable. 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 below [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, the temperature of a machine, the starting of processing in a machine, the termination of a process, or breakdown message about a machine. The amount of temporal data generated is thus, huge. The manufacturing data may be composite type. For example, the data representing position of a vehicle is a set of two coordinates.

2.4 Information retrieval

In order to control a manufacturing operation, an accurate and prompt feedback on actual performance is required [John87]. The temporal data is stored as it is generated. Information is extracted from it at a later stage. The status data 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 analyse 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 analysing 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 utilitization 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 databases and the information derived from it is converted into a format easily understood by the user. In the situation where the production process is automated, a graphical display of the process aids the operator in controlling the plant.

Chapter 3

The Object Oriented Modeling

A study of the domain revealed that the manufacturing data has some special features. A few examples are cited here to show the type of relationships among the data. A product may be an "assembly" of sub-assemblies. The tool-magazine "contains" a number of tools. A transporter "can be" programmed or manually driven. The Bill of Materials (BOM) for a product is obtained by recursively exploding its subparts. Traditional models such as the relational model cannot adequately capture all the above relationships. Recent research has resulted in the emergence of the object oriented data model which has a natural representation for these relationships. Hence, we have adopted the **object oriented** approach for modeling the data in the manufacturing domain. This approach to data modeling views the real world as being made up of a number of objects [Booc94, Rumb+91]. The *object model* is a collection of objects in the domain under consideration, which describes their static characteristics, and the interrelationships among the objects.

In this chapter, we describe the concepts involved in the object oriented approach to modeling and how relationships between the objects in the manufacturing domain can be represented naturally using this technique. Later, we describe the stages of development for the MIDAS system using the Object Methodology Technique (OMT) [Rumb+91].

3.1 The object oriented model

Some of the concepts characteristic of the object model are *objects*, *object identity*, *object class*, *aggregation*, *inheritance*, *encapsulation*, and *polymorphism* [BM91, Rumb+91, Booc94].

Objects and Object Identity:

Every real world entity is modeled as an object which is associated with a unique identifier. The object identity, or the OID, distinguishes the different objects in the domain. The OID ensures that the each object is unique by its inherent existence and not by their descriptive properties. An object has attributes to describe its state and methods which characterise its behaviour.

Object Class:

An object class represents objects with common structure, behaviour, relationships to other objects and semantics. An object class is an abstraction which helps to generalise the characteristics of individual objects. An object in the domain is an instance of the object class. Material handlers, fuels, raw materials are examples of classes. An AGV, a barrel of diesel, a consignment of pig iron are respectively, instances of the above classes.

Aggregation:

An object class could have attributes which are other object classes. The aggregation feature captures the containment or "a-part-of" relationships between object classes. Aggregation causes a nesting which results in a tree of objects called the aggregation hierarchy. Aggregations could be

- Fixed, that is, the number and types of subparts are predefined. For example, the accounts ledger class contains exactly two types of entries, account receivables and account payables.

- Variable, that is, the aggregate hierarchy is of a fixed level, but the number of parts may vary. For example, the tool magazine may contain one or more tools. This containment is a one level aggregate hierarchy.

- Recursive, that is, the number of aggregation levels is unlimited. The Bill of Materials for product is a common example for a recursive aggregation.

Inheritance and Class Hierarchy:

Inheritance captures the "is-a" relationship between object classes. The inheritance feature is unique to the object model. This feature distinguishes it from the hierarchical model. This mechanism permits classes to inherit features from existing classes. These derived classes called subclasses inherit all the attributes and methods of its parent class. A subclass may be derived from more than one parent class and can override features of its parent class. The ordered set of parent classes and their subclasses is called the class hierarchy. The manufacturing domain involves a number of "is-a" relationships. For example, material handling could be manually done or could be done with a transporter. Transporters may be programmable or manually driven. The programmable ones could be further distinguished depending on the flexibility of their motion.

Encapsulation:

This feature is important during implementation of the class. A class is encapsulated if its internal structure is hidden from other classes. The internals of the class are accessible only through the methods of the class. Encapsulation facilitates reusability of code and interfacing with the objects of the class becomes easier.

Polymorphism:

Polymorphism means that the same operation may behave differently on different classes. This feature is termed overloading in modern programming languages. Functions as well as arithmetic and unary operators can be overloaded. In function polymorphism, different routines can have the same name and the context is used to decide which routine is to be invoked. In the case of overloaded operators, the symbols could have a different meaning for different classes. The methods of a super class can be interpreted differently by each of its subclasses. This is implemented by defining in the subclasses a method with the same name and a different implementation. For example, the *replenish_stock()* method of the *Material_Input* class could mean different actions for objects of the *Raw_Materials* class and the *Sub_Assembly* class. In the first case, it could automatically trigger a purchase order for that item but in the second case, it could cause a production order to be released. The different types of polymorphisms have been described in [CW85].

The above features of the object model aid in portraying the manufacturing domain entities accurately. Having decided the means of representing the objects in the domain, we now describe the various stages in the development of the MIDAS system.

3.2 Building the MIDAS System

We have developed MIDAS system using the Object Modeling Technique (OMT) presented in [Rumb+91]. The OMT methodology suggests an object-oriented approach for software development. There are five major stages towards development of MIDAS, namely, *Analysis of the Problem*, *Data Modeling*, *System Design*, *Object Design*, *Implementation*. We briefly describe each of the stages of development of MIDAS.

1. Analysis of the Problem

   This is the first stage of development of the system. The manufacturing domain was studied and the characteristic features of the domain were analysed. The requirements of the system were identified and the objective was stated in the form of a problem statement.

2. Data Modeling

   This stage is concerned with devising a concise and correct model of the manufacturing domain. The model describes the various objects in the manufacturing world and their interrelationships. Construction of the object model involves the following steps:

   - *Identifying the objects and classes*
   - *Identifying associations (including aggregations) between objects*
   - *Identifying attributes of objects and links*
   - *Organising object classes using inheritance*

- *Verify access paths for likely queries*

3. System Design

   In this stage, the architecture of the system was designed. The system architecture refers to the overall organisation of MIDAS. The approach to implementing the system was finalised. This stage involves making the choice of the backend engine most suitable for MIDAS. The details of the graphical user interface and the interface with the backend engine were worked out.

4. Object Design

   The object design provides a detailed basis for the implementation stage. This stage is necessary to adapt the model to the database. It determines the definition of the classes and the associations used in implementation. Design Optimisation involves the following steps.

   - Adding redundant associations to minimise access cost and maximise convenience
   - Rearranging computation for greater efficiency
   - Saving derived attributes to avoid recomputation

   Implementation details such as implementation of associations and aggregations were worked out. The exact representation of object attributes were determined.

5. Implementation

   This step converts the proposed model into software. The data model was finally implemented in the database. The event handling mechanisms of MIDAS were incorporated and functions were added to access and manipulate the database objects. The GUI to the MIDAS system was built finally.

Chapter 4

Analysis of the Problem

This is the first stage of development of the MIDAS system. In the analysis stage, we analyse the nature of the domain, its features and identify the requirements of the system we intend to design. This analysis is presented formally in the form of the problem statement.

4.1 Statement of the problem

The goal of the MIDAS project is to develop a database system design tuned to support flexible manufacturing systems. MIDAS, as required of a DBMS, performs data management and thus provides a common data storage and retrieval facility. Additionally, since it lies in the control loop of the FMS, it actively participates in management of manufacturing processes.

There are three issues to be considered while designing the system.

- Nature of data of the manufacturing domain
- Transactions for the manufacturing domain
- Desired features of the MIDAS system.

4.1.1 Nature of domain data

The manufacturing data, as described in section 2.3, is complex and voluminous. The magnitude of the number of interrelationships among the different data objects can be seen from the MIDAS object model [Appendix C, Appendix D]. There are numerous instances of generalisation among the various objects. Tools, containers, fixtures, and palettes are essential for a manufacturing process. These objects have common features and may be grouped under accessory equipments. There are many instances of aggregation or containment relationships between the objects of the manufacturing domain. A few of them have been listed here. A manufacturing cell "contains" processing units, sensors, accessory supplies and material handlers. Bill of materials or parts explosion is another example. A process or operation performed on the workpiece may be regarded as a sequence of subprocesses.

The temporal nature of some objects of the manufacturing scenario raise the important issue of the large volume of data updates at every instant. Parameters such as temperature and pressure may have a significant impact on the quality of the product and may have to be monitored very closely. Lot tracking information must also be maintained in the database.

Summarising the characteristics of manufacturing data, we see that

- The volume of data handled is very large
- The data is complex by nature and has a lot of interrelationships
- The status component of the data rapidly changes over time

It is necessary that the controller is abreast with the status of the system, so that it can make suitable control decisions. This implies that the controller must have immediate access to the structural, status, and control data. It is proposed in the MIDAS model, that instead of the controller explicitly reading the temporal data and taking suitable actions, the data itself be linked to the action to be taken. A suitable mechanism is required to associate the action with the data.

4.1.2 Transactions for the manufacturing domain

The transactions for the manufacturing environment are different from conventional applications such as banking, and accounting. The transactions in these applications require serializability to ensure correctness. The scenario is different in the manufacturing environment. Different transactions require different isolation levels.

The status data is constantly changing. This data may be archived so that later, a temporal history of the data can be obtained. There are two distinct groups of transactions which access the sensor data, the "updaters" and the "readers". The updaters are plant monitors while readers are MIDAS processes. The updaters work in separate data partitions since they update a different set of plant variables. When the updates are crucial, the transactions need to adhere to the serializability and correctness issues. Updates of status information such as sensor values are independent of the current value of the data object. These updates are called blind writes [EN94]. The isolation level for the transactions depend on the nature of the update or read. Depending on the requirements and functionality, transactions of lower degrees of isolation can be used to increase concurrency [GR93]. For example, a transaction collecting statistics may be permitted to read values which committed after this transaction was initiated. On the other hand, transactions reading and writing sensitive data such as fault information or accounting must confirm to the serializability and correctness.

The structural data may be read or written by the plant operators and the MIDAS processes. Inventory updation, customer authentication, etc. are executed on structural data. These transactions must ensure correctness.

The control data can be read and written by plant operators or by MIDAS processes. The control settings may be either under operator control or automatic control. It is necessary that only one process be able to update the control variables at a time. Hence, concurrency control is not required in this case. However, the transaction construct is necessary to ensure that the update is made completely.

4.1.3 Desired features of the MIDAS System

After analysing the data and transactions of the domain, we specify the features of the system to be designed. In this section, we have discussed the desired features of MIDAS with respect to various issues such as data model, nature of transactions, recovery techniques, real-time response, decision support, control, and schema evolution.

- Data Model and Transactions

  Since many of the manufacturing objects are complex, the MIDAS system must be capable of supporting complex objects. This implies that the MIDAS system must support composite attributes as well as relationships such as inheritance and aggregation.

  The nature of transactions on the different data items are predictable and known. Lowering isolation levels for transactions which are sure not to introduce significant inconsistencies in the implementation will increase concurrency. Hence, the MIDAS system must also support the different isolation levels for transactions.

- Frequency and Volume of Updates

  As described earlier, the manufacturing database is characterised by extremely frequent updates of the temporal data. This volume of data collected in a given time interval is large even for small plants. The factory status data, which is continuously changing, describes the current state of the concerned manufacturing element. This temporal information is sent by the plant monitors. Since they refer to different plant variables, their updates in the database usually do not interfere with each other. These updates are frequent and large in magnitude since the status of the plant changes continuously. Hence, MIDAS must be capable of catering to voluminous updates.

- Fault tolerance and Recovery Techniques

  It is intended that MIDAS handles management of data as well as online control of the plant. Hence, in the event of a MIDAS system failure, MIDAS is no longer a part of the control loop. The recovery procedures of MIDAS must not only bring the underlying DBMS engine to a consistent state, but it must also communicate to the factory elements to prepare them to resume control over manufacturing. These recovery mechanisms are specific to the factory concerned.

- Real-time response

  The factory monitoring tools and sensors continuously feed data into MIDAS. This data is read and depending on the rules apriori defined to the system, the appropriate expert or decision systems are referred to and control messages are sent to the shop floor. A control instruction issued to the equipment controller might be time critical. Hence MIDAS must consider the real-time aspect to the transactions.

- Decision Support and Embedded Control

  MIDAS must have control embedded into it. The data actively communicates with the the control unit of the system and notifies it of significant changes in the data value. Control routines that help plant control must be an integral part of MIDAS. In order to help the operator evaluate the potential impacts of different control decisions, MIDAS must also have integrated into it expert systems for decision support. When more than one action is possible, the expert system routines decide on the best of the available options after considering values of relevant plant variables.

- Schema Evolution

  New manufacturing objects may have to be introduced into the database. This could be due to introduction of a different variety of equipment, or due to inclusion of a new performance parameter. MIDAS must be capable of accomodating changes in schema.

The features of the problem domain and the goals of the MIDAS system have been stated. Our aim is to incorporate into the working prototype as many of these features as possible. In the next stage, we describe the object model.

Chapter 5

Designing the Data Model

The design of the data model involves identifying the objects, their various classes, their attributes, the relationships between them, and verifying access paths for likely queries. In this chapter, we describe the MIDAS object model. We have used the Rumbaugh [Rumb+91] notation, given in Appendix A to pictorially represent the classes, associations and inheritance. The MIDAS Object Model has been presented in Appendix C and Appendix D.

5.1 Identifying objects and classes

The manufacturing domain comprises of numerous classes. Following is an enumeration of the various classes that have been identified.

The *MIDAS_Object* is a generic class, representative of all the objects in the domain. The *Manufacture_Object* and the *Personnel* classes represent objects in the manufacturing domain and the manual workforce involved, respectively. The *Production_Unit* class represents the highest logical unit of production within a structured manufacturing plant. The *Production Unit* class may be a *Simple_Cell* class or a *Compound_Cell* Class. *Operational_Inputs* class corresponds to the those accessory items required for a manufacturing system to enable processing. The *Processing_Unit* class is the generalised representation of a machine or of a manual production center where operations are carried out on the workpieces. *Operations* class represents the operations. The *Equipment* class represents all mechanical, mechanised, automated tools and equipments. *Tool_Magazine*, *Container*, *Tool*, *Fixture*, and *Palette* classes represent tool magazines, containers, tools, fixtures, and palettes respectively. The monitoring equipments are represented by the *Sensors* class. The *Material_Handler* class as the name suggests, is representative of all the devices used to transport materials from one point to another.

The *Storage* class corresponds to the places where semi-finished workpieces, products, raw materials and other such items are stocked till they are retrieved at a later stage. The *Long_Term_Store* and *Transient_Store* classes respectively represent the stores which hold goods for longer and shorter durations of time. *Long_Term_Store* class specialises into a *Warehouse* class or a *Cell_Store* class. *Short_Term_Store* class specialises into the *Equip_Buffer* class or the *Item_Buffer* class.

The *Hardware* class represents all the hardware equipment used to run the software which is represented by the *Software* class. *CPU* and *Peripherals* class represent the details of the CPU and other peripheral equipment. *Production_Software* class refers to the programs which are directly used for production. *Ancillary_Software* class is concerned with the software tools for decision support and other such functionalities. These tools are not directly involved with control and are aids for the controller. *Operating_System* class refers to details of the underlying OS. The *Controller_Unit* class represents the equipment controller and the *Supervisory_Control* class represents higher level control software such as the cell level and factory level controller.

The *Workpiece* class represents the workpiece prototype. This may be a raw material at the beginning of the process, a semi-finished part, or a finished product. The *Lots* class represents the lots or batches of semi-finished parts. *Product* class refers to the products that are manufactured in the plant. *Design_Details* class represents the details of the designs of instances of the *Manufacturable* class. *Material_Input* class refers to the materials which are required to produce the products. These may be instances of the *Raw_Materials* class or the *Sub_Assembly* class. The *Manufacturable* class represents the subassemblies which can be manufactured in the plant. The *Customer*, *Vendor*, *Subcontractor* classes denote respectively customer, source of purchase for factory, and subcontractor to do a particular stage of the processing. The *Order* class represents the order placed by the customer for one or more products. *Purchase_Order* class represents the purchase order placed for the raw materials and other supplies by the factory.

The *Process_Sheet* class refers to the process plans or the list of processes corresponding to the production of a particular workpiece. The *Route_Card* class represents the route cards that are associated with every lot. The difference between the process sheet and the route card is that the former lists the sequence of operations that have to be done and the list of processing units which can perform that operation whereas, the latter lists the details of operations on the lot by the different processing units. The *Move_Wp* class represents details of movement of the workpiece. The *Move_Lot* class refers to the actual details of how the lot moved on the shop floor. *Storage_Details* class corresponds to storage specifications for the raw materials, products, semi-finished parts, supply equipments, and operational inputs. *Lot_Store_Det* class is representative of the actual storage details of a particular lot.

5.2 Identifying aggregations

There are many examples of containment relationships among objects in the manufacturing domain. They are discussed below.

- The production unit is the highest logical unit of a production environment. It may be a simple cell or a complex cell. A simple cell is a manufacturing cell which comprises of a set of automated machines and a material handler. A complex cell is a collection of cells which may be simple or complex. There are material handlers in a complex cell which cater to inter cell part movement. This is shown in Figure 5.1(a). This is a recursive aggregation.

- The simple cell comprises of one or more processing units, supply equipments, sensors, and material handlers. This is shown in Figure 5.1(a).

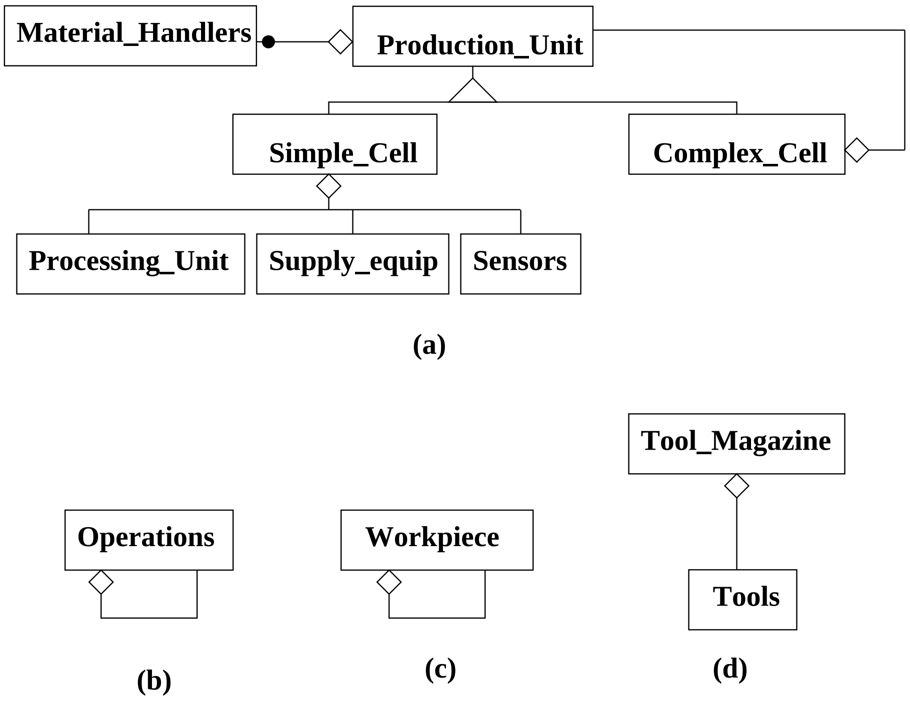


Figure 5.1: Aggregation in MIDAS object model

- A single logical operation may contain one or more sub-operations. For example, drilling a hole on a part may involve several steps like orienting the part, setting the drill at the coordinates, drilling the hole, and cleaning the part. During process planning, only the higher level abstraction of the process is considered. Depending on the nature and importance of the sub-operations, details of the sub-operations may be stored. Figure 5.1(b) shows this aggregation. This is a recursive aggregation.

- A workpiece prototype may comprise of a combination of more than one workpieces. This is a recursive aggregation and is shown in Figure 5.1(c).

- A tool magazine contains one or more tools. This aggregation is shown in Figure 5.1(d).

5.3 Identifying other associations

The classes of the MIDAS object model have a number of relationships between them. These associations may be binary, ternary or higher order associations. In our model, we have used only binary and ternary associations. Qualified associations [Rumb+91] are explained subsequently.

- Binary Relationships

  The following relationships are some of the binary associations used in the model. They may be one-to-many, many-to-many or optional-to-one associations.

  The storage units have the capability to store a workpiece-prototype.

  They may be currently holding lots of different parts.

  The long term storage units belong to a simple cell or complex cell.

  A transporter, automated or manually driven, may be used within a long term storage unit.

  Equipments may be stored within a long term storage unit.

  A workpiece prototype may be material input or a product or scrap.

  The items in a lot must be a type of workpiece.

  A subassembly may be a product.

  A customer places orders.

  An order may be for one or more suborders.

  Purchase orders contain requirements of raw materials.

- Ternary Relationships

  The following relationships involve objects of three classes.

  Process_Sheet : A Process Sheet contains the process plans or the sequence of operations and the possible processing units which can perform that operation corresponding to the production of a particular workpiece. It involves the *Operations*, *Processing_Unit*, and the *Workpiece* class (refer A in Figure 5.2).

  Route_Card : The Route Card contains the list of processing units actually visited and the details of operation start and end times. Hence, this involves the *Operations*,

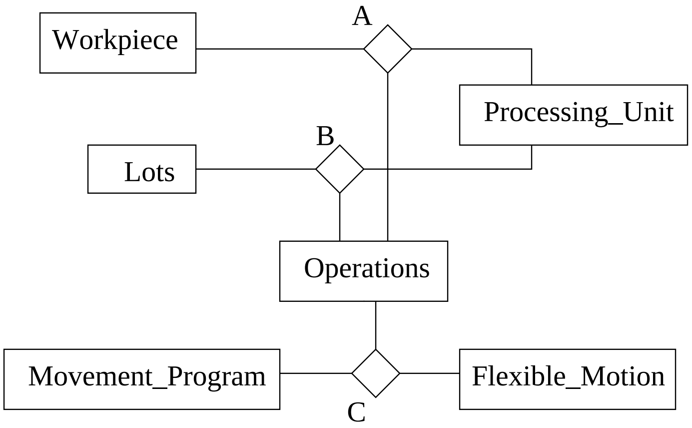


Figure 5.2: Ternary Relationships in MIDAS object model

Processing_Unit, and the *Lot* classes (refer B in Figure 5.2).

Move_wp : A workpiece can be transported by a material handler. The movement itself is a type of operation. Hence, it involves the *Material_Handler*, *Operations*, and the *Workpiece* classes.

Move_Lot : The details of how a lot was transported and the type of movement are represented by this class. It involves the *Material_Handler*, *Operations*, and the *Lot* classes.

Transport-Operations-Program : This link describes how a transporter belonging to the *Flexible_Motion* class performs an operation using a part program. It involves the *Flexible_Motion*, *Operations*, and the *Movement_Program* classes (refer C in Figure 5.2).

Purch_Detail : A purchased lot relates to the vendor and the purchase order. Hence, this association involves the *Vendors*, *Purchase_Order* and the *Lot* classes.

- Qualified associations

  A qualified association relates two object classes and a *qualifier*. The qualifier partitions the set of related objects into disjoint subsets, the subsets comprising of one or more objects. This relationship is analogous to the weak entity set relationship in relational design [EN94]. For example, every product design comprises of several drawings. A drawing name is unique for the design, but may be reused in other designs. In other words, the objects of the *Drawings* class are uniquely identified by the drawing sheet number and the design to which they belong. The sheet number is the qualifier. Similarly, every manufacturable subassembly has several designs associated with it. The objects of the *Design_Details* class are uniquely identified by a combination of the design number and the subassembly that it describes.

5.4 Identifying attributes of objects and links

Attributes help to describe the state of an object. The super class contains attributes and operators common to all its subclasses. The subclass contains additional features which distinguish it from other subclasses of the super class. The inheritance relationships between object classes in the model are explained later. *Link attributes* describe the associations between object classes. In this section, we discuss the attributes of some of the important object and link classes in the model. We begin the discussion with a few of the object classes.

- Production_Unit Class

  The attributes of the *Production_Unit* class are unit number, set up date, layout details, and environment details.

- Order Class

  The objects of *Order* class are uniquely identified by the order number. Attributes of the order class are order date, priority number, status, final payment date, date of dispatch, delivery date, bill cost, penalty clause, and terms of purchase.

- Material Handler Class

  Attributes of this class are vehicle ID, capacity, speed, route, and maintenance log.

- Machines Class

  The *Machines* Class is different from the other classes in that, it inherits properties from two classes, *Equipment* class and *Processing_Unit* class. An equipment has a name, a function, storage specifications and maintenance logs. Processing unit is described by the nature of operations, number of personnel required, and production capacity. Performance parameters such as defect rate and utilization are used to evaluate a processing unit. A machine has additional features such as machine description and details particular to the type of operation it is capable of performing such as shape of the spindle and palette size.

Some associations may be modeled as classes. In this case, the link attributes become the attributes of this class. Alternately, the attributes of the link may be stored in one of the associated objects. Implementation of associations have been explained in section 7.2. Described below, are some of the associations which were modeled as separate classes.

- Suborders Class

  The *Suborders* class contains the specifications of the Customer Order for each product, that is, a suborder object is created for every item in the customer order. Attributes of this class are Order number, Product code, quantity, and specifications.

- Lot_Store_Det Class

  An object of the *Lot_Store_Det* class is distinguished by the lot number and the storage unit ID. This class represents the exact location of a lot in the storage, if the workpieces are stored in lots. Attributes of this link class are arrival time, storage position, removal date, and expected removal date.

- Storage_Details class

  Objects of the *Storage_Details* class are distinguished by workpiece code and the storage unit ID. The storage unit could be either short term or long term storage. This class stores the details for storage of a workpiece prototype in the store. The link attributes are cost of storage, reorder level, maximum level, and storage life.

- Process_Sheet class

  The *Process_Sheet* class has the following attributes: operation stage number, standard cost, duration of operation, cushion time, number of personnel required, besides the candidate key consisting of processing unit ID, operation number and the workpiece code. This class stores information about the sequence of operations a workpiece has to undergo. This information is used during scheduling and capacity planning.

5.5 Organising object classes using inheritance

There are a number of hierarchical relationships in the manufacturing domain. The *Production Unit* as explained earlier represents the highest logical unit of production. This cell may be a *Simple Cell* or a *Complex Cell*.

A workpiece is processed at the processing unit. The processing can be done either manually (represented by *Manual_Prod*) or with machines (represented by *Machines*). *Machines* may be *Operator_Driven* or *Program_Driven*. The *Lubricants*, *Coolants*, *Cleansers*, *Chemicals*, and *Fuels* classes inherit common properties from the *Operational_Inputs* class. The *Machines*, *Sensors* and *Supply_Equip* are subclasses of the *Equipment* class. It is interesting to note here that the *Machines* class inherits properties of both the *Equipment* class and the *Processing Unit* class. This was necessary since a machine can be viewed from two perspectives, as a processing device or as a piece of equipment. This is shown in Figure 5.3(a).

The *Storage* Class may be categorised into *Long_term_store* and *Transient_store* classes. The *Warehouse* and *Cell_Store* are subclasses of the first type and *Equipment_buffer and Item_buffer* classes are subclasses of the second type. The different items in the shop floor may be stored in the warehouse, which is a common long term store for the entire plant or in the cell store, which caters to requirements of the cell. The workpieces wait for their turn for processing in a temporary buffer. The empty palettes, containers, etc. wait in the equipment buffer. It is possible that the same buffer is used for both the fixtured

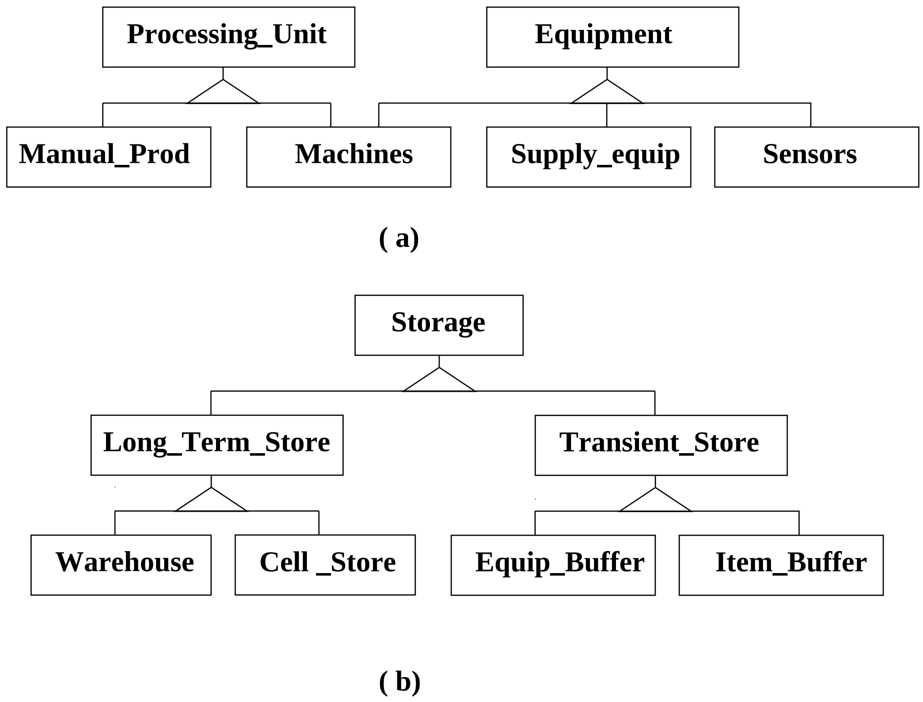


Figure 5.3: Inheritance in MIDAS object model

workpieces as well as the empty palettes and containers. This hierarchical relationship is shown in Figure 5.3(b).

Supply_Equip represents the family of accessory devices and tools represented by the *Containers*, *Tools*, *Fixtures*, and *Palettes* classes. *Sensors* are devices designed to monitor the status of the plant or the machine. They may belong to either of *Environ_Monitor* or *Production_Monitor* class. The former measures the environmental conditions. The latter may be *Defect_Detector* which detects defective workpieces and products or *Counter* which merely counts the number of pieces. This is shown in Figure 5.4(a).

The *Material handler* class is at the root of a deep hierarchy. A material may be moved manually or with a transporter, hence the *Manual move* and *Transporter* classes. The Transporter class, in turn, may belong to the *Programmable* or *Manual Driven* class. The former may be further categorised based on the type of movement. The *Flexible_Motion* class represents subclasses like *AGV*, *ASRS*. The *Fixed_Motion* class may be classified into subclasses *Conveyor*, *Gantry_robot*, and *towline_cart*. The Manual_Driven Class includes

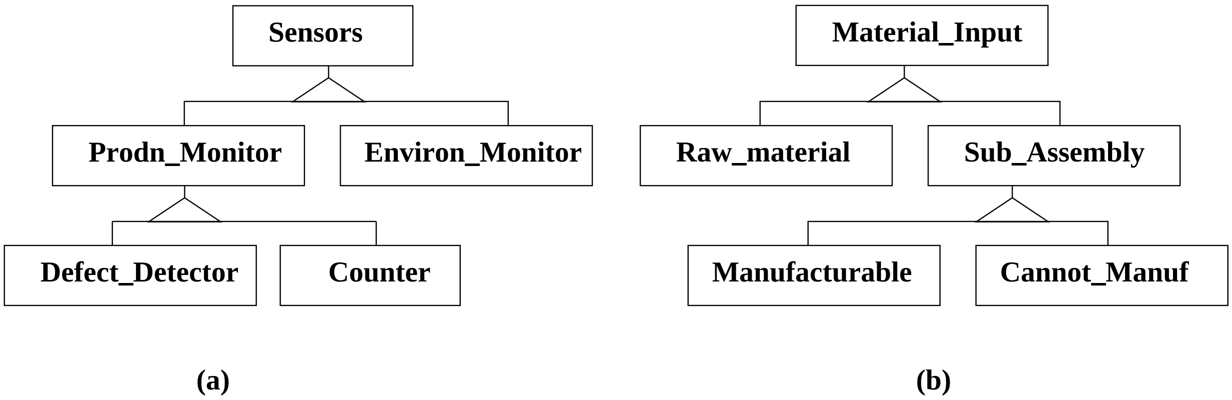


Figure 5.4: Inheritance in MIDAS object model

subclasses like *Cranes, Trolleys, and Forklift* classes that represent transport devices which have to be driven manually.

The *Material_Input* class can have subclasses *Raw_Material* and *subassembly* classes. The latter class may be further categorised into *Manufacturable* class which can be manufactured in the factory and *Cannot_Manuf* class which cannot be manufactured in the factory and have to be purchased. This is shown in Figure 5.4(b). The *Product* class may be sub-divided into *Standard_Product* and *Customised_Product* classes. The *Standard_Product* class represents the the standard products marketed by the factory. The *Customised_Product* class represents the products which were customised to the requirements of a customer.

5.6 Verify access paths for likely queries

This is the final step of the modeling stage. Here, we trace access paths through the object model for various queries. The access paths may traverse one or more associations. Consider the following queries:

- *Fetch the status information for a machine W in a specified time interval.*

  The status information of the machine in the given time interval can be obtained from the *Machines* class. The lots which were processed at that machine and the operation that was performed on it can be obtained from the *Route_Card*. This data must be formatted and displayed to the user.

- *Display the contents of all pending Orders of Customer X.*

  The query is processed by obtaining the order number from the *Customer-Places-Order* link. Now since we have obtained the order number, the product code and quantity of each product for every order can be obtained from the *Suborders* link class.

- *List the Bill of Materials for product Y.*

  A workpiece is a product at its final stage. First, the query determines the workpiece prototype corresponding to the product. Then, it recursively fetches the sub parts or subassemblies the part is made up of using the self-aggregation at the *Workpiece* class.

- *Fetch the temporal history of operations on the Lot Z.*

  This information can be obtained from the *Route_Card* association class. This class stores the lot number, operation, start and end times of the operation and machine on which the operation was performed. Information about where each lot was stored can be obtained from the *Lot_Store_Det* class.

Flow of messages

There are a number of messages exchanged between the objects during manufacturing. The controllers of the machines and the material handlers communicate with cell controller to receive control instructions and and to send status data information. All the decision making in the cell is done by the cell controller.

Consider the messages generated when the operation on a part has just completed and the part is to move to the next processing unit. The cell controller directs the machine to unload the part to the output buffer and start loading the next part. It reads the process sheet and directs the material handler to load and move the finished part to the next processing unit's input buffer. The status data for the processing units would be the change in machine state from PROCESSING to UNLOADING. The material handler generates status data for indicating the state change through the LOADING, TRANSPORTING, and UNLOADING states. This data is temporal information of the processing units and the material handlers. The status data for the lots involve the time of completion of the operation and the loading, movement, and unloading times at the material handlers. This information is entered into the route card of the lot.

Summarising the contents of this chapter, we have identified the various objects of the manufacturing domain, established relationships among them and tested the object model with typical queries for access paths. The next stage is to decide the system design.

Chapter 6

MIDAS Sytem Design and Architecture

All the objects of the manufacturing domain and the relationships among them were identified and represented in the object model. After the Modeling stage, it is necessary to select a suitable system to represent the model. The principle of MIDAS design is to have the factory operator interact solely with the MIDAS database. From the operator's perspective, the database embodies the manufacturing system. Any input from the operator to the manufacturing system or from the factory to the database must be made through MIDAS. The actions to be taken as a consequence of the data update are effected through the embedded control feature of MIDAS. The system design must be such that it is possible to implement features of MIDAS which were described in section 4.1.3.

The organisation of this chapter is as follows. First, we discuss why the existing databases are not sufficient for implementing features of MIDAS. We then make a detailed specification of the requirements of the backend DBMS. This is followed by a discussion on the MIDAS system architecture.

6.1 Need for new database design

Current database technology does not satisfy the requirements for implementing a data management and control system for an FMS. The following discussion is on why traditional systems are unsuitable for developing a manufacturing database.

- Modeling power

  Traditional DBMSs usually implement a relational data model. The type of applications that may be developed on them are restricted to those whose domain can be modeled using the relational model. The Manufacturing database comprises a variety of objects such as text, design drawings, spatial information, and part sub-assembly description. These are complex objects and share different relationships such as aggregation, inheritance, and associations. which can be naturally represented in a OODBMS. The traditional DBMSs are built on simple abstractions and the mapping of a complex object is done by flattening its natural hierarchy.

- Lack of decision support facilities

  Standard off-the-shelf DBMSs are structured for traditional applications such as banking, accounting, etc. Hence, it is not surprising that most of these DBMSs lack functionalities for decision support. The controller of a manufacturing system often consults the expert systems for decision making for various activities.

- Support for active response

  Most conventional DBMSs are primarily geared towards applications that do not require the database to react automatically when a particular data value changes. The manufacturing database that we desire must support active features like triggers which are necessary for factory control. These triggers are fired with change in data value.

The software available today cannot be directly applied for controlling a manufacturing system. One solution to this problem would be to build a new system specially tuned for data management and control of an FMS. Another solution is to build MIDAS on top of available tools. It was decided that the MIDAS architecture would comprise a backend DBMS which would furnish all the requirements of MIDAS as a DBMS. Additionally, functions would be incorporated to MIDAS to give it the functionality of a controller for an FMS. Since MIDAS requires embedded control, the database must be capable of actively responding to "events" in the system.

The first step towards building MIDAS is identification of features of the back-end engine. The next section identifies the requirements of the backend for MIDAS.

6.2 Features required of the backend DBMS

A DBMS forms the backend engine of MIDAS. Other features which give MIDAS the functionality of a controller are built around this DBMS. The following are the features expected from the backend DBMS.

- Data model: The manufacturing domain comprises a number of complex objects. For example, a manufacturing cell contains a number of machines, monitoring sensors, and equipment such as tools and fixtures. The factory layout details require spatial objects to show position. The position of an AGV is represented by three coordinates. These objects need to be represented naturally in the database. Hence, the DBMS must support features such as complex objects, inheritance, aggregations, etc.

- Homogeneous interface: The system must present a uniform interface to all the other plant software modules. This makes the task of integrating the system easier. This interface must be independant of the database formats and structures of the individual sub-systems in the FMS.

- Fault tolerance and recovery: It is necessary that when the DBMS recovers from a crash, it must reset the database to a consistent state. Some of the recovery techniques used by conventional DBMS are checkpointing and logging. Since the database has an active role in controlling the system, a system crash results in loss of temporal data.

- Knowledge management: The system must be capable of supporting rules and policies of the factory. The designer must be able to associate actions with these rules. This feature will enable the the DBMS to provide active response to database events.

- Setting different consistency levels: Unlike the conventional applications like banking, all the transactions in the manufacturing environment need not satisfy the ACID properties. Different transactions require different isolation levels.

  For example, discrepancies in factory status information such as data for statistics will not affect the correctness of execution of the controller. Failure to register updates will only affect analysis. Under overload conditions, it must provide real-time response to the important sensors and must selectively ignore those of lower priority.

- Support for legacy data: Currently all manufacturing data is stored in relational or hierarchical format. It must be possible to suport legacy applications with the new system.

- Extensibility: MIDAS essentially comprises of a database engine which can be extended to include the set of control functions that enable it to control the plant. This is possible only if the database engine provides an application interface for the user (system designer) written programs.

We have listed the requirements of the backend engine on top of which MIDAS can be developed. Next, we discuss the architecture of MIDAS.

6.3 MIDAS architecture

The MIDAS architecture consists of three layers.

- Illustra backend engine
- MIDAS server
- MIDAS server interface

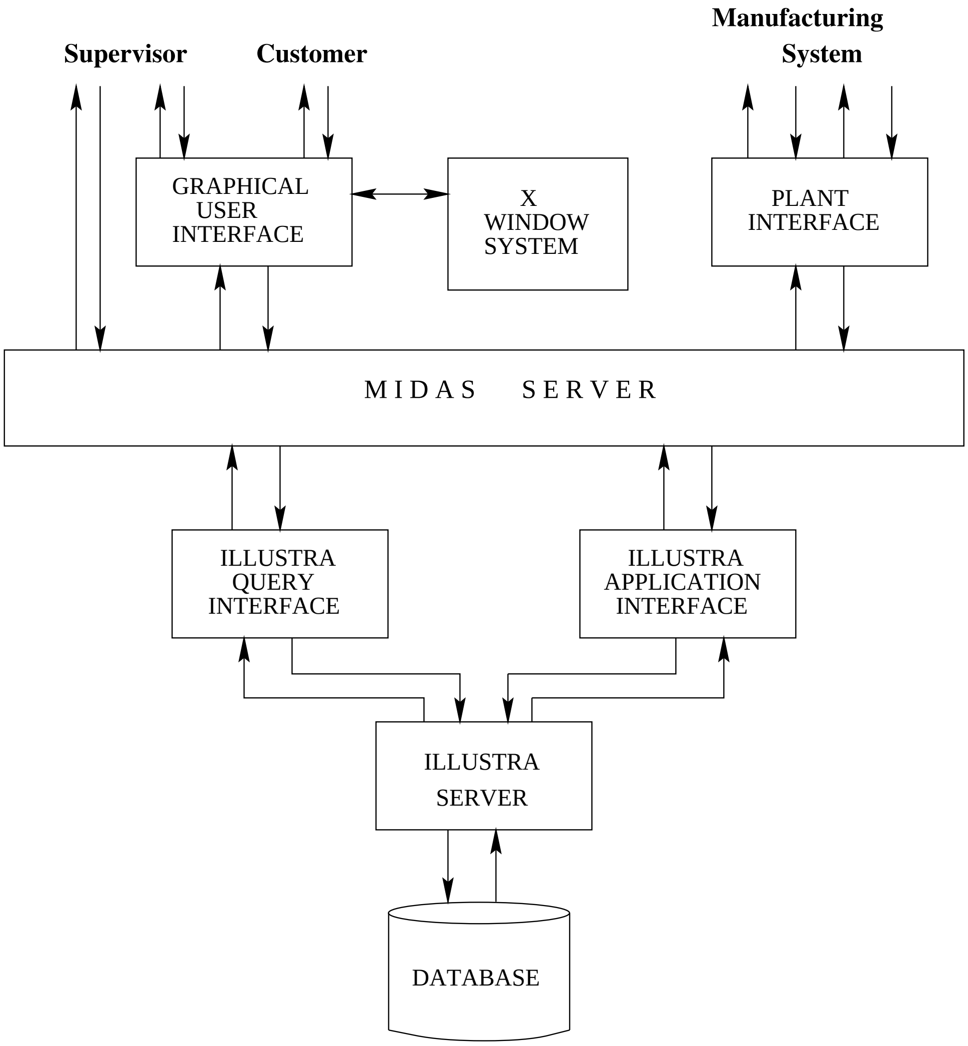


Figure 6.1: MIDAS Architecture

In the following subsections, we discuss the three layers of MIDAS.

6.3.1 Illustra backend engine

Illustra is a commercially available Object-Relational DBMS. It is essentially a relational DBMS supporting object oriented features complete with a query language and access methods [IlUG, IlAPI, IlDAG]. Illustra supports Standard Query Language (SQL) for accessing the database. It provides an elaborate set of library routines which may be used for developing applications. The Illustra DBMS serves as the backend engine for MIDAS.

Most legacy applications used for the manufacturing processes are based on the relational model. Illustra maps the objects into tables in the database. Hence, adapting the legacy application to the MIDAS database becomes easier. Thus, this combination of relational and object oriented approach is appropriate for our system.

Architecture of Illustra

Illustra uses the client server architecture for handling the data of the database [IlUG]. Illustra processes comprise

- midaemon, which is the main Illustra server process
- Elan License Manager, which manages licence issues for Illustra products
- miserver process, one instance of which is spawned for every client process
- client process, which could be the query interface or the application programming interface

Illustra features

The Illustra DBMS provides a number of features which are not present in most traditional databases. The following discussion is on the those features of Illustra, which are not common in other commercially available DBMS.

- Complex Objects: Illustra is basically an extended relational DBMS. It provides a OO interface to the underlying relational model. Every object corresponds to a table in the database. Composite attributes of an object are stored in separate tables which are automatically created by Illustra. Composite attributes may be arrays, sets or a combination of two or more attributes. There is no support for member functions of an object.

- Functions: The user can create functions which may be defined in the SQL syntax or a C function. Functions can return a value or a set. They can be used to build iterators. It is possible to bind operators to functions and to overload functions.

- Rules: The rules system in Illustra is event-driven. A rule can be specified for only one object class which may be the entire table or a column of a table. Illustra uses the *Event-Condition-Action* concept for handling events in the system. When a *Database Event* occurs, it takes an *Action* after evaluating the *Condition* which is based on the rules which have been apriori specified to the system.

- Isolation Levels for reads: Illustra supports four levels of isolation for reading data.
  - Read Uncommitted: This level permits transactions to read uncommitted values of the data.
  - Read Committed: This level permits transactions to see data as it is when the transaction started.
  - Repeatable Read: This level does not permit transactions to read uncommitted transactions but it does not protect against phantom reads.
  - Serializable: This is the highest level of isolation where the transaction does not see the results of uncommitted transactions and serializability is maintained.

- Alerters: Illustra has a provision by which it can raise alert signals. These are used by application programs for information of occurrence of asynchronous events.

- 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.

Drawbacks of Illustra

The features of Illustra seem suitable for our application. However, Illustra is not without its drawbacks.

- Though Illustra logically supports objects, they are implemented in the database as tables. Hence, during schema evolution the designer is consciously aware of the relational method of storing data.

- Just as in other OODBMSs, Illustra does not provide a natural method of supporting aggregation relationships. Aggregations were represented as associations and sets in the implementation.

- An application program which is "listening" for an alerter, has to poll for the signal. This implies that the the listening process spends useless CPU time waiting for the alert. A possible workaround is to assign a process exclusively for polling for the alert message. This polling process communicates the occurence of an alert signal.

- When updates of temporal data are made to a table, due to the no-overwrite feature, the size of the table grows quickly. Hence, at frequent intervals, data which has become stale must be moved to another storage.

The inadequacies of Illustra make the task of the MIDAS designer more difficult. The Illustra DBMS no doubt has its failings, but the features it offers such as support for complex objects, ability to support overloaded functions, the support for rules and alerters, and time travel render it suitable for use in a manufacturing system.

Illustra as a backend engine

The backend engine forms the lowest layer of MIDAS. Illustra receives requests for various operations on the objects in the database. It obeys rules which have been previously specified to the system. It fires "alerters" which notify the listening application of the occurrence of an event. Illustra is responsible for concurrency control, database recovery,

transactions, and storage management.
Illustra offers two interfaces to the users:

- Interactive Query Interface: This interface may be invoked through MIDAS. It may be used by the system designers and developers during schema evolution and schema modification in the database.

- Application Programming Interface: This interface includes a rich set of library functions. The application program calls appropriate Illustra routines to access the data stored in the database. It also includes direct access library routines which bypass the query parser, rule system and the optimizer. The direct access routines are intended for use in performance-critical routines. The software modules of the MIDAS server call appropriate Illustra application programming interface routines to access the data stored in the database.

6.3.2 MIDAS Server:

The MIDAS server ideally, should be an intelligent, decision-making DBMS capable of domain control. Logically, MIDAS comprises of the Illustra backend engine, a set of rules or decisions and software routines for factory control. The Illustra backend engine was discussed in the earlier section.

MIDAS exploits the "active" feature of the backend engine to provide real-time control over the plant. Rules may be specified in the backend engine or through separate software routines or by a combination of both. When changes or updates are made to the data, the rules determine the function to be invoked. The control routines which actually evaluate the plant status and send control signals are the same as those used in the regular FMS controllers. The difference here is that they are invoked by the DBMS. The decision making modules may be expert systems which have been integrated into MIDAS. Currently, MIDAS does not incorporate expert systems.

For example, when a lot of inventory is removed from the stores, the current_quantity attribute is decremented. The "rule" alerts the supervisor if the stock is below the minimal mark. Or when a "job-over" event is reported from the factory, the "rule" fires a trigger which calls the control routines module to send an appropriate control instruction to the machine.

6.3.3 MIDAS Server Interface:

There are two interfaces to the MIDAS server, the interactive interface and the plant interface. The interactive interface is a Graphical User Interface (GUI) developed using X and Motif [Youn90], [Motif].

Interactive Interface

The interactive interface provides two levels of interactions.

- Customer Interface : The Customer Interface is a very restricted view of the plant. It provides information of interest to a customer. It contains views of products, their description and designs details of the product. The Customer Order Entry and enquiries regarding the Customer Orders can be done through this Interface.

- Operator Interface or Supervisory Interface : The Operator or the Supervisory Interface on the other hand, provides a detailed view of the manufacturing process. It displays the entire layout of the plant under supervision. Configuration details of the equipment and its current status can be obtained by clicking on the appropriate process icon. There is a control panel display to enable the operator to control the plant. The operator can change the control settings of the equipment through this interface. For example, a machine can be disabled or a lot could be removed from or included into the process by the operator. The operator may also override the decisions made by the FMS.

  View Generation

  The temporal data must be presented in a format that can be easily understood by the operator. This information is obtained by appropriately combining status data. The Interface also provides three views, a plant view, a process view and a lot view [Taka+96]. These views are provided for a time interval specified by the user. They help the operator get a detailed view of the entire process.

– Plant View

  The Plant View provides a view of the entire plant. It represents *"which batch was being produced in which processing unit"* in the time interval. On specifying a time interval, the view displays a Gantt Chart which shows the lots which were at each of the processing units during that time interval.

– Process View

  The Process View provides information about a particular processing unit in the plant. It represents *"how the process status was when each batch was being produced"* in the time interval. Given a time interval and a processing unit, the view shows up all the lots which were processed in that processing unit, and the sensor data during that interval. This view helps the operator to analyse the process status for each batch. This view also helps to determine how the quality of a product is affected by variation in the sensor data of the processing unit.

– Lot View

  The Lot View provides the view point of a specific lot out of the final products. It represents *"how the status was at every process when the lot passed through"*. This interface also helps in lots tracking. The operator can trace the history of the lot and its ancestor lots. On specifying a lot number, the view displays processing information of this lot and of the ancestral lots which make up this lot. This view is particularly useful when the cause of defect of a product lot is to be determined. The history of production for ancestors of the concerned lot can be studied and inferences may be drawn.

There is also a provision in the interface to bypass the GUI and interact with the database. In this case, MIDAS simply connects the user to the Illustra Query Interface. This interface is mainly intended for the designer of the system. It may also be used by the intelligent supervisor to query for details not available through the GUI. This has been listed under this category since it provides unrestricted access to the user.

Plant Interface

Different types of messages pass between the Equipment level controller and the MIDAS Server. These messages are one of

- Event messages indicating "job done" or "buffer full" from the plant to the MIDAS Server.
- Requests from the plant for operational data such as part program to the MIDAS server.
- Control instructions from MIDAS Server to the appropriate equipment controller. These instructions are decided after interpreting the event messages and the configuration of the system.
- Operational data that were previously requested and which have to be downloaded to the equipment controllers.

Summarising the contents of this chapter, we identified the requirements of the backend DBMS and discussed the MIDAS system design. In the next chapter, we discuss the next phase of development of the system.

Chapter 7

Object Design and Implementation

The object design stage provides a detailed basis for the implementation stage [Rumb+91]. In the design stage, we discuss the design of associations, aggregations, and inheritance relationships. This stage helps to interpret the relationships in a DBMS independent form. In the implementation stage, the object design is mapped to the database schema using the DBMS Data Definition Language(DDL).

7.1 Design optimisations

The object model captures the logical information about the system. The design optimisations involve the following steps.

- Adding redundant associations: Redundant associations do not add any information to the structure of the object model, but are added to increase efficiency of access. They also introduce the additional burden of maintaining consistency. Access patterns and the frequency of access must be considered before including a redundant association. In our model, for the queries considered, we did not find the necessity to add redundant associations. Hence, we have not used any redundant associations in our model.

- Rearranging execution order: In the earlier step, the structure of the object model is altered to optimize frequent traversals. In this step, the algorithms for traversal of associations are optimised. For example, suppose there was a query for the names of all the customers who have ordered for a particular product. One way of retrieving this information would be to select the *Sub_Orders* objects for these products, obtain the Order number, then query for the *Customers* who placed these orders. The idea is to narrow down the search as much as possible.

- Saving derived attributes: Derived attributes are those which can be computed from other data. These attributes may be stored in their computed form to avoid the overhead of recomputation. They are updated when the base objects on which the derived data depend change. For example, the *stock_avail* attribute of the *Standard_Prod* class is the sum of the quantities of the product stored in each of the warehouses. This is a frequently monitored attribute and it would be more efficient to store this value along with other details of the class. Similarly, the *utilisation* attribute of the *Processing_Unit* class is derived from the schedules or may be computed periodically. It need not be recomputed everytime the processing unit's details are queried.

  Derived attributes must be updated when the base value changes. This could be done in one of the following ways.

  - Updating the derived object when the base object changes

    When the base value changes, the derived object may be either explicitly updated or updated by triggers fired by the updated base value.

  - Periodically recomputing the derived attributes

    Recomputation of all derived attributes periodically may be more efficient than incremental updates when derived attributes depends on several base attributes. This avoids the situation of the derived attribute being computed each time any of the base values change. On the other hand, if the changes in data values are few in number, this approach would mean recomputation of the entire set of derived values even though only a few are affected.

These decisions cannot be generalised and depend on the context of the data being updated.

7.2 Design of associations

Associations are the links in the object model which provide a means of navigation between the different objects. Depending on the requirements of the application, the choice of strategy for implementing an association is made. Associations may be unidirectional or bidirectional. The associations must be implemented such that access time for queries is optimised.

7.2.1 One-Way associations

If an association is always traversed in one direction, it may be implemented as an attribute containing an object reference. The class which contains the reference is the one from which traversal is initiated. If the multiplicity is "one", the attribute is a simple pointer. If the multiplicity is "many", then it corresponds to a set of pointers. Qualified associations may be implemented as one-way associations. The following are some of the associations modeled as one way associations.

- *The association between Workpiece and Product, Workpiece and Material_Input, etc. are modeled as attributes in Workpiece.*
- *The association between Manufacturable subassemblies and Design_Details is an attribute in Manufacturable class.*
- *The association between Design_Details and Drawings is an attribute in Design_Details.*
- *The association between Lots and Workpiece is an attribute in Lots.*

7.2.2 Two-way associations

Most associations in the manufacturing database are two-way associations. Two-way associations are those that are traversed in both directions. They may be implemented in three ways.

- The association may be implemented as an attribute in one direction and a search is performed when a backward traversal is required. The backward traversal is expensive since the entire set of objects must be scanned. This approach is useful if frequency of traversal in one direction is much more than in the reverse direction.

- The association may be implemented as attributes by both the objects involved in the association. Traversal is equally efficient in both directions. In this case, additional maintenance is required while updating the database. If either attribute is updated, the other must also be updated to maintain consistency. This approach is useful when accesses outnumber updates. It permits quick access along the association.

- The association may be implemented as a distinct association object, independent of either class. An association object is a set of references to the associated objects. In this case, we need to search all the objects of this association class for forward as well as backward traversals. Accesses are slower than in the previous two cases. This approach is preferred for establishing relationships between predefined objects since the association objects can be added without adding any attributes to the original classes. Distinct association objects are used to model sparse associations.

The choice of implementation depends on the nature of the data and the queries. Listed below are a few examples of some two-way associations in the MIDAS object model.

- *The association between Order and Customer is implemented as an attribute in the Order class.*

- *The association between Machines and Tool_Magazine is modeled as attributes in both the classes.*

- *The association relating Tools and the Processing_Unit that a tool is currently in is modeled as a separate class.*

- *The Lot_Dependencies are implemented as a separate object class.*
- *The link between the Processing_Unit class and the Personnel class is represented as a separate association class.*

7.2.3 Link attributes

The link attribute describes an association. If the association is one-to-one, link attributes may be stored as attributes of either object. If the association is many-to-one, link attributes may be stored as attributes of the "many" object. If the association is many-to-many, the association is implemented as a distinct class. In this case, each instance represents a link and its attributes. The following are instances of link attributes in the MIDAS object model:

- *The many-to-many association between Order and Product has attributes such as quantity and specifications. This association is modeled as a separate Sub_Order class.*
- *The many-to-many association between Workpiece and Storage has attributes such as cost_of_storage, reorder_level, max_level, and store_life and is modeled as the Storage_Details class.*
- *The many-to-many association between Lots and Storage is described by attributes such as arrival_date, storage_position, removal_date. This association is modeled as the Lot_Store_Det class.*

7.2.4 Ternary associations

Ternary associations are implemented as separate association classes. The ternary associations in the MIDAS object model also have link attributes. They have been listed earlier in section 5.3 on identifying associations. The *Process_Sheet*, *Route_Card*, *Move_wp*, and *Transport_Operations_Program* classes show an association between more than two classes and also contain other information about this association.

7.3 Design of aggregations

Aggregations are a form of associations which show the containment relationship. Implementation of aggregations is similar to that of associations. The aggregation may be implemented by embedding the contained object, or a reference to the object in the container object. For example, the *Simple_Cell* contains a set of references to the *Processing_Units*, *Supply_Equip*, and *Sensor* objects. For some queries, it may be required for the contained objects to know which object they belong to. Alternately, aggregations may be implemented as Two-Way associations as in the case of self-aggregation in the *Workpiece* and *Operations* classes.

7.4 Representation of object attributes

The attributes mentioned below describe the classes they are contained in. An attribute may be an atomic data value or composite value. For example, atomic attributes such as code number, name, and quantity are of integer, string, and numeric types respectively. Attributes such as *Maintenance_Log*, *Layout*, *Route Description*, and *Terms_of_Purchase* give a broad level view of what is to be represented. *Layout* attribute could represent an image of the entire plant. *Route* of a material handler could be represented as a sequence of machines or cells. It may also be represented as a series of coordinates corresponding to locations within the plant. *Maintenance_Log* could be represented as a collection of text. *Description* may be implemented as text or a collection of specifications which may be of a composite type. For example, *description* attribute for a product of a hot strip mill would include length, mass, thickness etc. [KR95], but this may not be true for a factory manufacturing pagers. Hence, the implementation of these attributes depends on the factory for which MIDAS is to be customised.

7.5 Implementation of MIDAS

The entire model was mapped to the the MIDAS database using the Illustra SQL DDL commands. Illustra offers the "setof" data type which helps to embed a set of pointers in an object. These are used to implement aggregation relationships. After the static objects and the associations were implemented in the database, the other features of the MIDAS system were incorporated. The Btree indexes provided by Illustra were created for often accessed objects. Various functions and routines for plant control were incorporated into the system. Rules and triggers were created and associated with different actions.

In Illustra, every object class maps on to a table. Here we illustrate with a couple of examples, how the classes were defined in the database. Through the first example, we show how inheritance and aggregation are implemented. The following statements define the *Tools* and the *Tool_Magazine* tables. One may recall that the *Supply_Equip* class is a subclass of the *Equipment* class. The *Tools* class is a subclass of *Supply_Equip* class. The *Tool_Magazine* class is an aggregation of objects of the *Tools* class. The aggregation has been implemented as an one-to-many relationship. A set of references to objects in the *Tools* class is maintained in the *Tool_Magazine* class. The declarations were made as follows.

```
--- Create the Supply_Equip table which is a sub class of Equipment
--- class
create table supply_equip
(       equip_id        char(25) not null primary key,
        equip_life      measures_t,
        life_left       numeric(10,2),
        maxlife         numeric(12,2),
        equip_descrip   text,
        material        char(25)
) under equipment;

--- Create the Tools table which is a sub class of Supply_Equip
```

```
create table tools
(       tool_name       char(25),
        capability       char(30)
) under supply_equip;

--- Create a user defined type with components same as the columns
--- of the table Tools
create type tools_t ( like tools);

--- Create the Tool_Magazine table
create table tool_magazine
(       toolmag_id      char(20) not null primary key,
        tool_type       char(20),
        no_of_pocket    integer,
--- The set of references to the tools it is currently holding
        holds_tools     setof( ref(tools_t)),
--- The tool magazine belongs to one machine
        belongs_mach    unit_no_t not null references machines (mach_id)
);
```

The next example is that of the *Sub_Order* link class. This class links the *Product* and the *Order* class and contains attributes such as quantity and cost. This link class references the primary key of the associated classes. The class declarations are as follows.

```
create table sub_orders
(       order_no        integer         not null,
        prodt_code      char(25)        not null,
        specifications  text,
        qty_ordered     numeric(12,3)   not null,
        cost            numeric(12,2),
--- The primary key is the Order number and the Product code
```

```
        primary key( order_no, prodt_code),
--- The columns order_no and prodt_code are primary keys of
--- Order( declared as prod_order) and  Product tables.
        foreign key( order_no) references prod_order,
        foreign key( prodt_code) references product
);
```

In the next chapter, we describe the simulation of a Coffee Making Plant, which uses MIDAS to manage production data as well as control its activities.

Chapter 8

A Manufacturing Plant Example

As a familiar example of a simple manufacturing plant, we have simulated the functioning of a Coffee Making Plant. It is assumed that the different machines in the simulated plant are automated and each of them are supervised by an equipment controller. A plant controller manages the activities of the plant and communicates with the machines through the equipment controller. This is a simple situation of a two level control system. This plant uses a database for data management. The example chosen illustrates how MIDAS can be customised to perform the role of a controller of this plant.

8.1 Description of the Coffee Making Plant

The example was developed from the details outlined in [Taka+96]. The automated Coffee Plant produces coffee powder from raw coffee beans. It blends the coffee powder in the desired ratio, strains the decoction, and mixes sugar syrup and milk to produce coffee. The sequence of the processes are detailed below.

- The coffee beans are roasted in the *Roaster*, one variety at a time.
- The roasted beans are ground at the *Mill* and the coffee grind is stored in a grinds tank.
- The *Blender* blends the coffee grinds in the desired ratio.

- At the *Dripper* the coffee is strained with hot boiling water. The grinds waste is sent to the grinds waste tank.

- The black coffee is mixed in the *Mixer* with sugar syrup and milk. The mixed coffee is then cooled and sent to the product tank. There are four flavours of coffee produced. The flavour depends on the sequence and the ratio in which the milk and syrup are mixed.

8.2 Choice of the example

The example chosen helps to show how issues such as lot management, status data collection, view generation, plant monitoring and control are handled by MIDAS. In this plant, the materials are transferred in units of lots, each lot identified by its unique lot number. Lots may combine to form a new lot or split into smaller lots. MIDAS stores in its database, the temporal history of the various processes and products in the plant. MIDAS also provides views of the plant which may be used for analysis of the plant processes. The example also provides scope for an on-line visual interface for the operator supervising the automated system. This example helps to show how plant control including handling of exception events can be incorporated into MIDAS. The activities of the plant are simulated and the events are communicated to the MIDAS Server.

Here, we discuss some of the aspects of the Coffee Making Plant example. The customer can place an order for any quantity of any of the four flavours. Only that amount of coffee that has been ordered is produced. The example incorporates product flexibility, that is, the product mix can be varied.

Manufacturing Process Control:

MIDAS controls the simulated plant. Status information about the processes namely, the temperature is collected periodically from the sensors. This information may be used by the controller to take corrective action when these values exceed the permitted range. This was implemented using the embedded control feature of MIDAS. The controller in MIDAS also manages the movement of lots. The lot movement data is recorded in the plant database. MIDAS also has an interface for the supervisor to intervene in the management of the plant.

Lots management:

This is a batch manufacturing plant. The coffee beans, the coffee grinds, the final product are produced and transferred in units of lots. The lots maintain their identity till they are mixed with other lots or are split into lots of smaller size. In order to be able to trace the history of a lot, it is required to maintain information about the ancestors of the lot.

Monitoring of Manufacturing Process:

The GUI interface to the plant helps in supervising the activities of the plant. The operator can also participate in the control of the plant through the controls panel provided as a part of the interface. The display of the interface is shown in Appendix B.

View Generation:

The quality of a product is influenced by the manufacturing process. The quality control manager might want to analyse the processes which a lot has gone through. The plant supervisor might want to know the status of the entire plant or might want to monitor a specific process. MIDAS therefore, provides three views for analysing the plant processes. These are the *Plant View*, the *Process View*, and the *Lot View*. These views have been explained earlier, in section 6.3.3. The views are generated from temporal status information stored about the various processes and from the lot management information. The cause for defects found in a particular lot can be traced back to the faulting process with the aid of these views. Thus, the views help in the analysis of the manufacturing process.

8.3 Implementation of the Coffee Plant model

8.3.1 Object classes used

The issues handled by the Coffee Plant example require only a subset of the object model for the manufacturing domain. Some of the important classes used were

- *Machines*

  The *Machine* class was instantiated into Roaster, Mill, Blender, Dripper, and Mixer for each of the processes. A simple machine controller was built to supervise the functioning of the machines. This controller guides the machines through the following states *IDLE, DOWN, LOADING, UNLOADING,* and *PROCESSING*. Temporal temperature information was stored for the *Roaster*, *Dripper*, and the *Mixer*.

- *Material_Handlers*

  This class was instantiated into AGVs and conveyors. These objects were controlled by the plant controller. A *Material Handler* could be in one of the following states, *IDLE, LOADING, UNLOADING, TRANSPORTING*, and *DOWN*.

- *Lot*

  The *Lot* class represented the different lot objects which were created for the raw beans and the blended mixture of coffee grind. These lots of raw beans were routed from the tank of beans to the coffee grinds tank. New lots were generated to represent coffee grind from the coffee grinds tank and to represent new lots output from the *Blender*. In order to provide the views, the lot movement data must be recorded in the database along with the time information.

- *Tank*

  The tanks which store the beans, grinds and the final product are instances of the *Tank* class. The tanks behave differently from the other buffers in that the contents of the tank are a homogeneous mixture, so the lots lose their identity.

- *Sensors*

  The temperature sensors used for determining the temperature are instances of the *Sensors* class. Their periodicity can be specified by the user. The sensor data shows the process status information. This is stored along with a time stamp.

- *Product*

  The four flavours offered by the Coffee Making Plant are four products of the plant. The products are instances of the *Product* class.

- *Customer and Order*

  The customers are instances of the *Customer* class. Every order the customer places is an instance of the *Order* class. An order may request for more than one product.

Some associations between the classes were implemented as association classes with link attributes. They are listed below.

- *Process_Sheet* shows the route the lot must take.
- *Route_Card* for a lot shows the start times and end times for each of the different processes the lot may go through.
- *Sub_Lot* stores the relation between a lot and its parent lots.
- *Sub_Orders* shows the orders for individual products.

8.3.2 Creation of a knowledge base

The above classes were created and incorporated into the database. Rules and alerters were then defined so that the DBMS responds actively to events in the database.

A sample set of the alerters that were created were

- *sens_roaster* to indicate temperature information about the roaster.
- *sens_dripper* to indicate temperature information about the dripper.
- *dummy_alert* which is a dummy alerter.

An alerter and a rule is declared in Illustra SQL as below:

```
create alerter sens_roaster  ( mechanism='poll');
create rule sens_roaster_more
on insert to sensor_info
where new.value > 80
do
alert  sens_roaster;
```

The rule specifies that when the sensor value is greater than 80 fire the alerter sens_roaster.

Each of the alerters were associated with appropriate functions to handle the event. The actions to be taken are specified by the user. The implementation of alerters in Illustra is such that the listening process must continuously poll for database alerts. The listening process recognises the alert and calls the appropriate routine. Since ours was a simulation study, we introduced a dummy alerter to indicate the last alerter of the alerters fired.

8.3.3 Tracing event sequences

On a "startup" signal from the operator, a message is sent to all the machines to get ready for processing. The machines convey a "ready" message to the MIDAS Server. MIDAS routes the lots of raw beans, grinds, and black coffee through the plant. The start and end times of every process are noted for the lots. When the machines finish one stage of processing, they send an "over" message to MIDAS. A control routine in MIDAS directs it to move to the next stage of processing. The sensors monitor the temperature at regular time intervals (the duration of the time interval is specified during configuration). The temperature is stored in the database along with the current time information. If the value is above or below the permitted range, the rules fire the appropriate alerter. The routine listening for alerters recognises the alerter and invokes the appropriate action. The implementation of functions by Illustra backend server is such that it is not possible to integrate an Illustra "C" function with the Coffee Making Plant simulation code. Hence, the actions cannot be directly invoked by the alerter. The actions are translated into appropriate instructions which can be interpreted by the plant.

8.3.4 Interface

The Interface was built using X and Motif. The interface interacts only with the database. Most of the queries from the interface use only read transactions. Hence, we use lower level isolations for the read transactions. However, for updates such as that from the operator, we use serializable transactions. Appendix B contains some snapshots of the display.

MIDAS interface offers an updated view of the entire plant. Configuration details of the equipment and its current status can be obtained by clicking on the concerned process icon. The details of the tanks and buffers can also be queried online. The operators panel helps to perform a limited set of actions, namely, freeze the plant status, discard a lot, stop a machine, start a machine.

Three different views of the status data are provided. The timestamp stored with every lot movement and with every sensor data help to integrate the data of the processes with the lot data and present a unified view of the plant activities.

1. Plant View

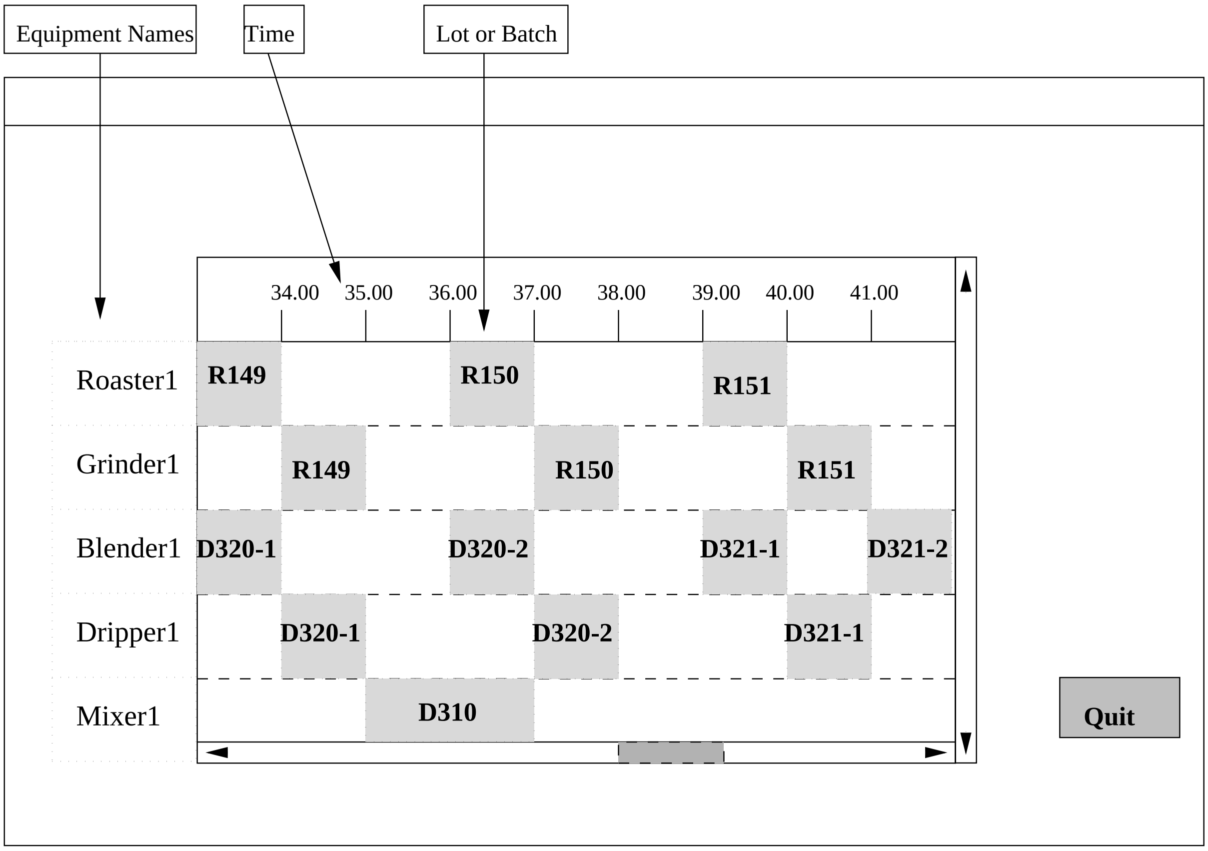


Figure 8.1: Plant View displayed on MIDAS

The Plant View displays the lots being produced at each process during a specified time interval. For generating this view, all the lots which were being processed within that time interval and the processing units which were processing them are retrieved. This information is displayed as a Gantt Chart. Figure 8.1 shows this view.

2. Process View

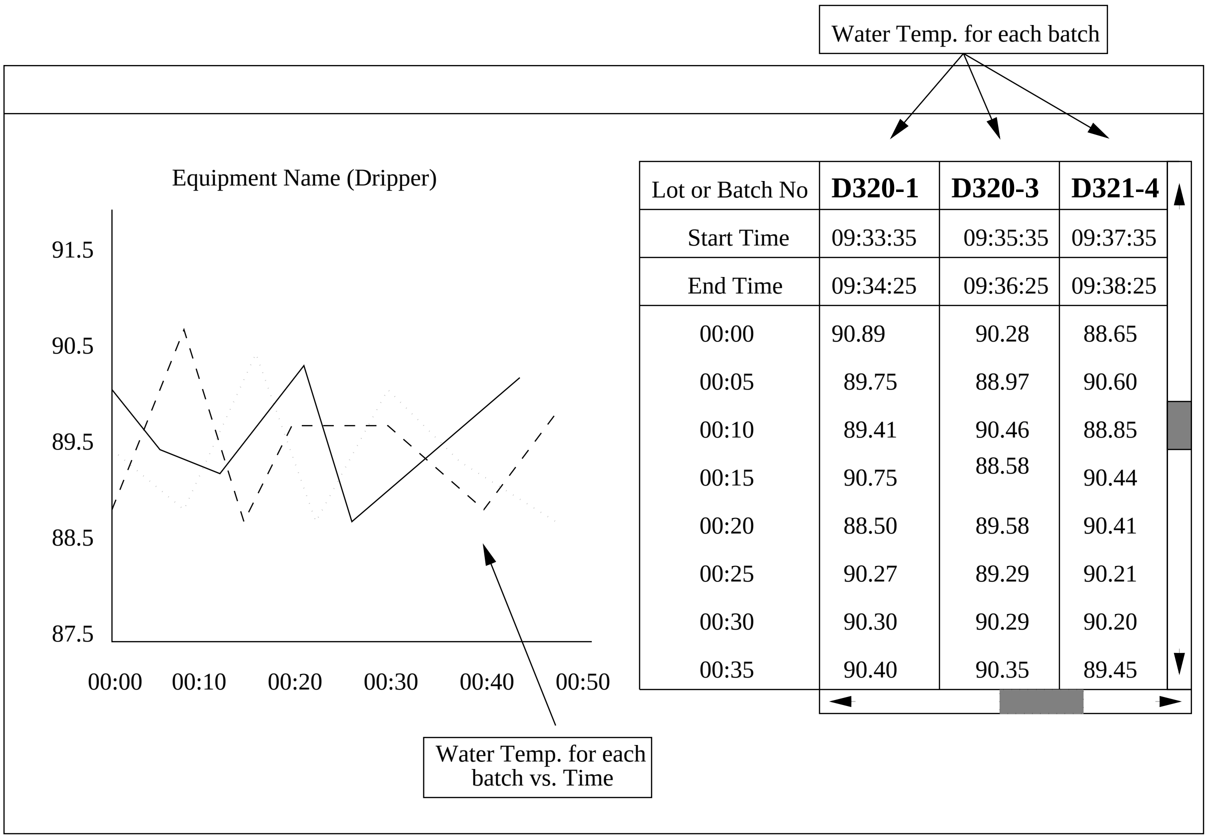


Figure 8.2: Process View displayed on MIDAS

The Process View represents the process status when a certain batch is being produced. For generating this view, all the lots which were processed in the time interval and corresponding status (here temperature) data of the process are extracted. The status data is displayed in the form of a graph. The X axis represents the time and the Y axis represents the temperature. Figure 8.2 shows this view.

3. Lot View

The Lot View represents the view point of one specific lot out of final products. This view helps to obtain the history of producing the lot from raw materials to the final product. This view is generated as follows. The processes and the process status data for every lot which forms the specified lot are retrieved. The status data is displayed as a graph with time along the X axis and temperature along the Y axis. Figure 8.3 shows this view.

8.3.5 Implementation details

A scheduler for the plant was developed to schedule the set of orders. The scheduling algorithm is based on priority dictated by slack time and the machine capacity [DJ90]. Each order specifies a time within which the order is to be fulfilled. The list of jobs to be scheduled are input to the scheduler. The jobs contain information about the product to be produced, the quantity, and the sequence of processes it must undergo. The scheduler fixes a priority for each job based on the available time left for delivery, the capacity of each of the machines and the jobs to be assigned to the machines.

A Coffee Plant Simulator simulates the events of a typical Coffee Making Plant. These events are communicated to the MIDAS server. Events such as "job complete" are interpreted by the MIDAS Server and instructions as to the next action of the machine are communicated to the corresponding machine. The data of these events was also recorded in the database for analysis purposes. The temperature values sent by the sensor are stored directly into the database through MIDAS Server. If the data value exceeds the permitted range, the rules fire the alerter. A Server function traps this message from the database and decides on the next course of action for the concerned machine. The GUI helps to interact with the plant. Besides providing various plant details, it also permits the operator to interfere with the plant working. Currently, the operator can startup or shutdown the entire plant, or start or stop a machine.

The entire object model had about 150 classes including association classes. In the coffee plant example, about 25 classes and an equal number of rules and triggers were used. About 20 Illustra functions were written. The example was written in C++. The code for the simulator and the scheduler involved about 9000 lines of code.

8.4 Adapting MIDAS

The Object Oriented approach to the modeling makes extensibility easier. In this section, we discuss how MIDAS can be customised to an automated plant.

The example of the Coffee Making Plant was not a true representative of an FMS. Nevertheless, it was sufficient to illustrate the principle of MIDAS design. The MIDAS database was made to synchronously reflect the actual plant status. This made it possible to develop the plant view generator independent of the plant control software. Hence, replacing and modifying the existing interface can be done without affecting the functioning of the plant. Additional views can also be easily incorporated into the system.

Extending MIDAS to accomodate new rules is easy. For example, to add another rule, the semantics of the rule have to be input to the database. If it involves firing an alerter, the alerter has to be created and the action of the rule is defined accordingly. In the routine which listens for the alerters, the new alerter name and the action associated with it are included along with the information about the rest of the alerters. The rest of the code remains unaltered. Modifying an existing rule simply implies that the rule has to be redefined in the database.

To an automated plant using a relational model, MIDAS offers the additional features of a better data model and an extensible view generation. Switching over from a relational DBMS to MIDAS is fairly easy. Most industrial applications use the relational model, which can be easily mapped into MIDAS. Some of the control routines which were initially part of the control software, may be incorporated into MIDAS through rules and alerters. The routines to interface the database with the control software can be incorporated easily through the MIDAS Server. Additionally, new tables may have to be constructed to accomodate the temporal data. Additional views may be incorporated into the system later. Legacy applications can be easily mapped to the Illustra database. Since MIDAS stores the temporal history of the plant's production details, plant monitoring and analysis becomes easier.

Customising MIDAS to an automated manufacturing plant, which does not use a relational database is not a trivial task. The entities of the plant must be identified and a model must be constructed along the lines of the object model of MIDAS. This model shares all or a majority of its classes with the MIDAS object model. It is reasonable to expect that the plant may require more specialisations or additional classes than what is provided by the MIDAS object model. Inclusion of these classes should be easy since their relationship with respect to the MIDAS model object classes can be easily determined. The class definitions and relationships are made for the classes newly introduced for this model. The classes are then mapped to the MIDAS database. Some of the control routines are incorporated as rules and triggers. As in the previous case, the rest of the controlling software are incorporated into the system. The routines for view generation are added later.

Information about the inclusion of new machines or resources must be available to the control software. Consider how the scheduler recognises the addition of a new machine. A new instance of the machine class is created. Information about the types of operations it can perform, and the operation times and cost are entered into the database. The scheduler considers the entire list of available processing units during scheduling. The information about this new machine is thus available to the scheduler. Thus, information about addition or deletion of resources is first made into the database which is later communicated to the control software.

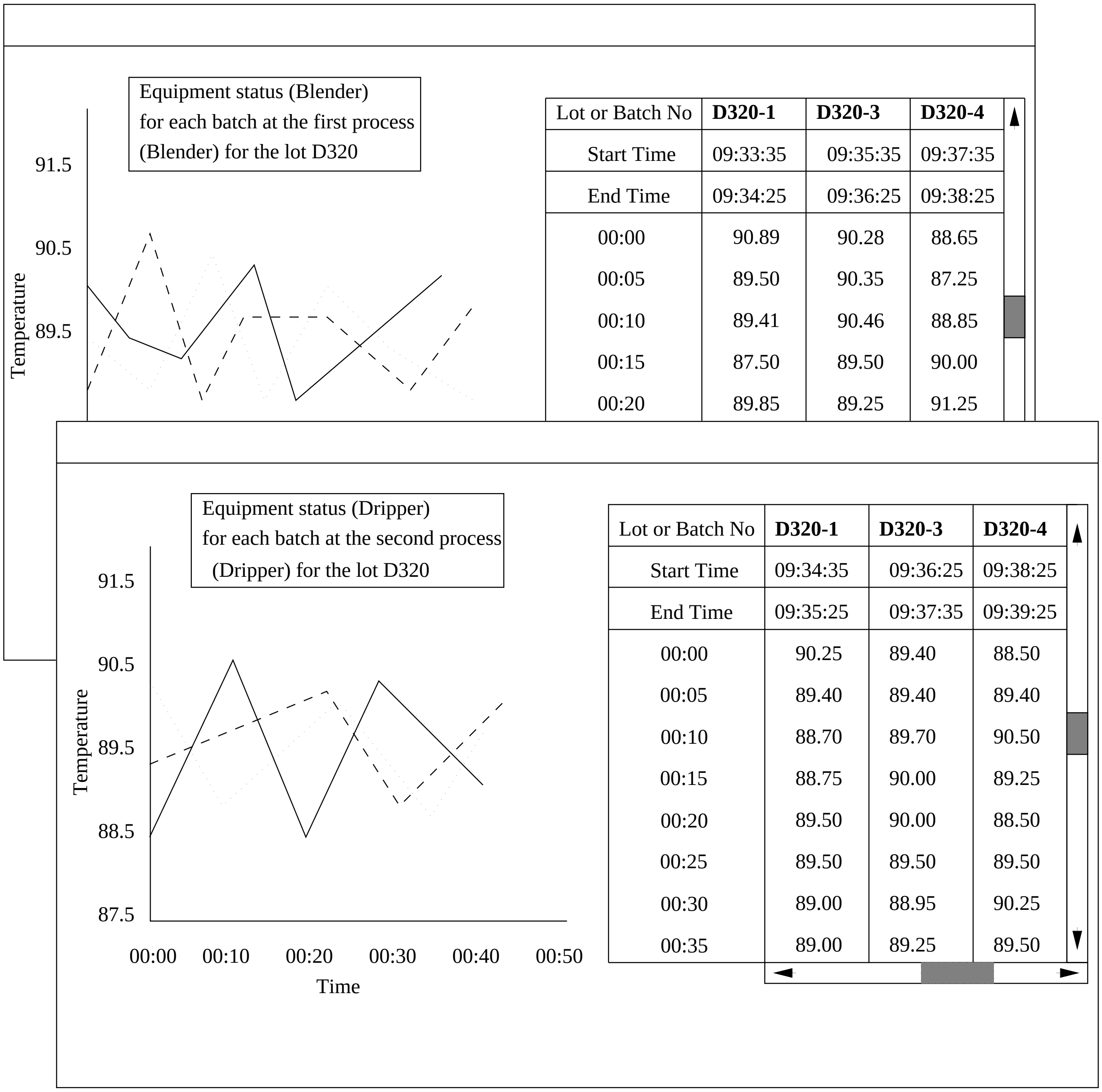


Figure 8.3: Lot view displayed on MIDAS

Chapter 9

Conclusion

9.1 Summary

Traditionally, the DBMS has always played the passive role of data storage and management. The current DBMSs are capable of features such as the object oriented data models and support for active response. The MIDAS System utilises these features for the design of a data management cum control system of the automated plant. Since traditional database models fail to meet the modeling requirements of the manufacturing domain, we have adopted an object oriented approach to designing the MIDAS System.

In this project, the manufacturing domain was studied and the objects of this domain were identified. The object-oriented approach helps to represent the real world objects naturally and hence, was used to model the manufacturing domain. The model was carefully analysed and mapped to a database schema. Since, MIDAS system should be able to support legacy applications, we use an object-relational DBMS for handling the data. The system architecture of the MIDAS System uses the Illustra object-relational DBMS as the backend. Currently, MIDAS incorporates control routines and a scheduler so that it can manage event-handling in an automated plant. In order to provide a detailed and a complete view of the different aspects of the plant, MIDAS offers an interface to the plant through the graphical user interface (GUI), which is an aid to the operator supervising the plant. By making the MIDAS database mimic the state of actual plant, we were able to isolate the view generator program and the control routines. The GUI interface interacts solely with the MIDAS database to retrieve and store information. MIDAS was tested through the example of the Coffee Making Plant. The example also helped to show the utility of the plant, process, and lot views during plant monitoring. Though the example of the Coffee Making Plant was not a true representative of an FMS, it helped to illustrate the principle of MIDAS design. MIDAS can accomodate inclusion and modification of resources as well as policies.

9.2 Suggested extensions

It was felt that there are a few issues yet to be considered, which would help to tune the system better to suit the real manufacturing world. They are discussed below.

- Currently, the MIDAS Server restricts itself to short term activities in the plant. The model can be extended to encompass all business activities including material requirements and capacity planning, master production scheduling, financial management etc. This would mean achieving an integrated data information system with a common database scheme.

- MIDAS handles control as well as data management. When MIDAS recovers from a crash, the recovery procedures of MIDAS must not only verify the consistency of the underlying DBMS engine but must also set itself to the factory status at the time of recovery. The MIDAS system has not been tuned for recovery from failure. Recovery mechanisms in the situation of MIDAS crash have to be worked out.

- The ability to track lots, and provide on-line information about lot movement may be extended to provide the customer continuous information about the status of his order. This would have significance when the plant deals with bulk orders for customised products.

- New efficient scheduling algorithms can be added to MIDAS. The user is thus offered a choice of scheduling policies. The user can select the one most suitable for his manufacturing environment.

- MIDAS may also be used as an aid for re-engineering the production system. The plant, product and lot views are a good means of analysing the system. Through these views it is possible for the plant operator to detect a faulty machine quickly. Also, the views from the temporal information maybe used to evaluate performance parameters such machine utilisation, and throughput. The plant view gives the supervisor an estimate of the waiting time between two operations for a lot. This may be used to study the behaviour of the on-line scheduler. Re-routing patterns for a lot in the situation of failure of the processing unit can be examined.

Bibliography

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

[BM91] E. Bertino, and L. Martino, "Object Oriented Database Management Systems : Concepts and Issues", *IEEE Computer*, April 1991.

[Blah88] M. Blaha et al, "Relational Database design using an Object Oriented Methodology", *Communications of the ACM 31*, 4 April 1988.

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

[CW85] L. Cardelli and P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism", *ACM Computing Surveys*, 17(4), 1985.

[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.

[DV90] P. Dobrivije and B. Vijay, *Distributed Computer Control for Industrial Automation*, Marcel Dekker Inc., New York, 1990.

[DW91] D. Dilts and W. Wu, "Using Knowledge-Based Technology to Integrate CIM Databases", IEEE Transactions on Knowledge and Data Engineering, Vol.3, No.2, June 1991.

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

[Forn+89] S. Forno, A. Fumagalli, and M. Morisio, "SDL to specify a Manufacturing System", *CAD/CAM Robotics and Factories of the Future*, Vol. 1, 1989.

[GR93] J. Gray and A. Reuter, *Transaction Processing: Concepts and Technology*, San Mateo, California, 1993.

[Groo87] M. Groover, *Automation, Production Systems and Computer Integrated Manufacturing*, 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, N0.6, December 1994.

[HB90] Hearn and Baker, "GRAPHICS", Wiley Eastern, 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.

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

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

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

[John87] J. Powers Jr., *Computer-Automated Manufacturing*, McGraw-Hill Book Company, 1987.

[Kore83] Y. Koren, *Computer Control of Manufacturing Systems*, McGraw-Hill Book Company, 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, McGrawHill, 1991.

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

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

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

[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.

[Shio+88] M. Shiojima et al, "Total Production Information Service System for the Factory", *CAD/CAM Robotics and Factories of the Future*, Vol. 2, 1988.

[Sptr93] "Manufacturing a la Carte", *IEEE Spectrum special issue*, September 1993.

[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.

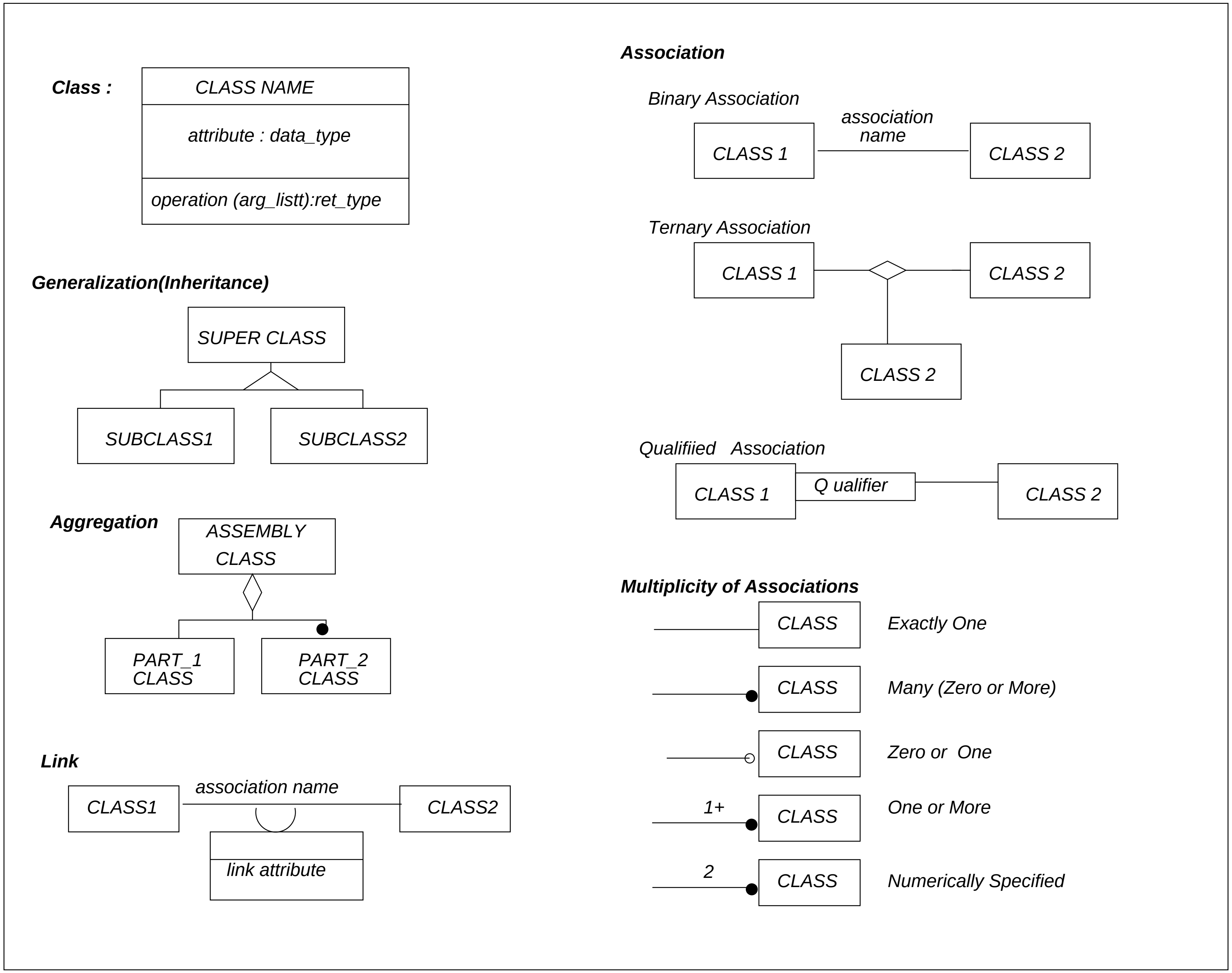
[Taka+96] H. Takada et al, "Production Information Management for Batch Manufacturing Plants Based on ECA Mechanism and View Generation", Unpublished manuscript, IESL, Mitsubishi Electric Corporation, Japan, 1996.

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

[Youn90] D. Young, *The X Window System*, Englewood Cliffs: Prentice Hall, 1990.

Appendix A

Rumbaugh Notation


Class :
CLASS NAME
attribute : data_type
operation (arg_listt):ret_type
Generalization(Inheritance)
SUPER CLASS
SUBCLASS1
SUBCLASS2
Aggregation
ASSEMBLY CLASS
PART_1 CLASS
PART_2 CLASS
Link
CLASS1
association name
CLASS2
link attribute
Association
Binary Association
CLASS 1
association name
CLASS 2
Ternary Association
CLASS 1
CLASS 2
CLASS 2
Qualifiied Association
CLASS 1
Q ualifier
CLASS 2
Multiplicity of Associations
CLASS
Exactly One
CLASS
Many (Zero or More)
CLASS
Zero or One
1+
CLASS
One or More
2
CLASS
Numerically Specified

Appendix B

Coffee Plant Display