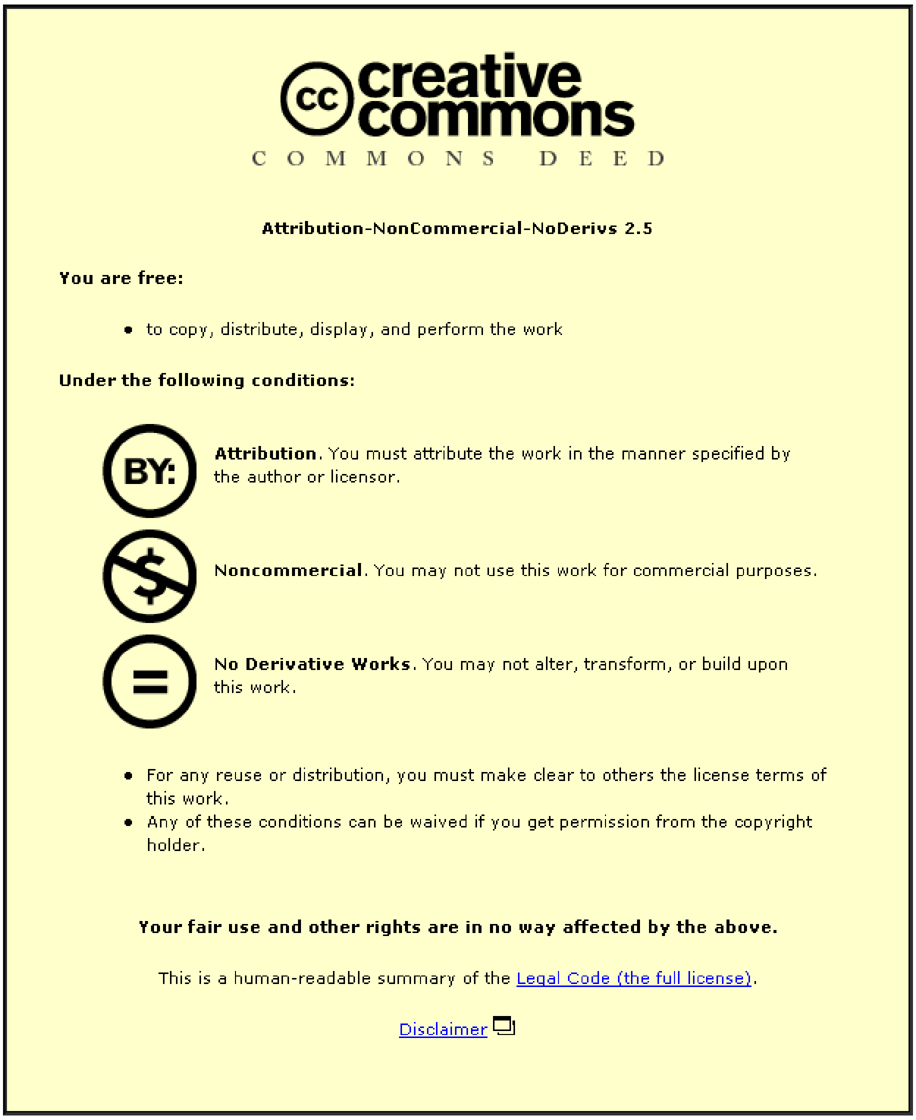


Applications Integration for Manufacturing Control Systems with particular reference to Software Interoperability issues

by

Valdew Singh

A Doctoral Thesis
submitted in partial fulfilment of the requirements
for the award of
Doctor of Philosophy
of Loughborough University of Technology

September 1994

Department of Manufacturing Engineering
Loughborough University of Technology

To :

My wife, Magdalene, for her love, support and encouragement and our children Sabrina and Sean for the constant joy they bring.

and

In loving memory of my late father, Banta Singh, who inculcated in me the belief in the pursuit of knowledge for self fulfilment and self-realisation.

CONTENTS

ACKNOWLEDGEMENTS

The author is grateful to Loughborough University of Technology for its financial sponsorship and the Economic Development Board (EDB) of Singapore for its support and kind understanding in granting study leave.

The author wish to express his sincere thanks to his supervisor Professor Richard H. Weston for his invaluable supervision, encouragement and help throughout this research study. Thanks must also be extended to my director of research Professor David J. Williams, and those staff of the Department of Manufacturing Engineering who have been kind and helpful. The author wishes to fully recognise the contributions, in terms of conceptual thinking of other members of the Manufacturing Systems Integration (MSI) Research Institute at Loughborough University of Technology. Particular thanks are due to Mike Leech, Paul Clements, Allan Hodgson, Jack Gascoigne, Shaun Murgatroyd, Ian Coutts, Awalludin Mohamad Shaharoun, Binglu Zhang, Marcos Aguiar and Frank Welz.

Finally, the author would also like to take this opportunity to express his gratitude to the following :

- Professor Alan Mulhleman and David Halsall of Bradford University Management Centre for offering case study material.
- Mr. Nick Telepneff of Manufacturing Systems Portfolio PLC (MSPL) for sharing his technical expertise and for supporting the SIM PCID product.
- Mr. Jon Whelan of John Brown Systems PLC for his technical support and training in the use of MCC production management application software.
- Mr. Peter Yeomans of MicroMatch PLC for sharing his experience and imparting his knowledge of the $IDEF_0$ and $IDEF_{1X}$ software modelling tools.

GLOSSARY

ACME
The Applications of Computers in Manufacturing Directorate of SERC (Science and Engineering Research Council.

Assemble-To-Order
A product where all components (bulk, semi-finished, intermediate, subassembly, fabricated, purchased, packaging, etc.) used in the assembly, packaging, or finishing process are planned and stocked in anticipation of customer order.

Bespoke system
A system designed and implemented specifically for a particular (one-off) application or site.

CAD/CAM
Computer-Aided Design/ Computer-Aided Manufacturing.

CAPM
Computer-Aided Production Management.

CAPP
Computer-Aided Process Planning.

CASE
Computer-Aided Software Engineering (tools) used to speed up and formulate the process of software design. Such systems use a variety of representations such as data flow diagrams, entity-relationship diagrams, and in some cases generate program code.

CIM-BIOSYS IIS
Loughborough University of Technology Manufacturing Systems Integration Research Institute's systems integrating infrastructure

CIM Model
A representation of a CIM system.

CIM-OSA
Open Systems Architecture for CIM - the output of a major ESPRIT project (No. 5288 : AMICE) which attempts to formalise the design, implementation and running of "open" CIM systems.

Configure-To-Order / Engineer-To-Order
Products whose customer specifications require unique engineering design or significant customisation. Each customer order then results in a unique set of part numbers, bills of material and routings.

Database
It is a mechanised, formally defined, centrally controlled collection of data.

Database management system (DBMS)
A software system which performs the functions of defining, creating, revising, and controlling the database.

Database prototyping
Technique used to discover the information requirements and to construct a data model.

Data integrity
The ability of the database to remain correct during operation.

Data Manipulation Language (DML)
Highly non-procedural languages associated with database system, e.g 4GL.

Data store
The set of data storage facilities containing the shared data.

DCE
Distributed Computing Environment.

Distributed database
Database spread across a network of computers.

DNC
Distributed Numerical Control.

Driver
In the context of this thesis a driver is used to enable the CIM-BIOSYS IIS to incorporate proprietary devices and applications which have their own specific protocols.

ESPRIT
The European Strategic Programme for Research and development in Information Technology.

Essential model
A primary model or representation (of some aspect of manufacturing systems, objects or process) at a generic level.

FIMM
FIMM (Functional Interaction Management Module) has been developed through this research effort with the purpose of tying together a set of applications into a coherent system to enable software interoperability. It controls and coordinates the interaction between applications and also synchronises the flow of activities within the system to support part manufacture.

IDEF
ICAM DEFinition methodology - a set of systems analysis and design tools.

Information of common interest (shared information)
Information which is created by one function and is often processed and/or used by several other functions within the enterprise.

Integration
The aggregation of resources and applications into a synergistic whole.

Integration toolset
A set of complimentary software programmes to assist in or enable some aspect of the development or management of CIM systems.

IRDS
Integrated Resource Dictionary System.

Life cycle
The specification, design, implementation and useful life of a system.

Legacy

The term legacy (systems, components and software) is used to refer to a previously installed base of systems, components and software. Legacy elements will not normally conform to the methods and standards which will be adopted in current generation solutions.

Make-To-Order

A product which is manufactured after receipt of a customer order. Frequently long lead-time components are planned prior to the order arriving in order to reduce the delivery time to the customer. Where options or other subassemblies are stocked prior to customer orders arriving, the term "assemble-to-order" is frequently used.

Make-To-Stock

A product which is shipped from finished goods, "off the shelf," and therefore is finished prior to a customer order arriving.

MCS

Manufacturing Control Systems: Specialised applications which exhibit discrete functionality for the rationalisation, improvement, control and execution of activities in support of part manufacture.

Model

A model can be defined as a tentative description of a system that accounts for properties relevant to the intended purposes of the model.

Normalise

The decomposition of more complex data structures into flat files (relations). This forms the basis of relational databases.

Open solution

A solution (to a CIM requirement) which does not constrain the user to specific proprietary hardware, software or protocols.

Platform (of integration services)

A software system which provides a consistent set of integration services (interaction, information and configuration) to manufacturing applications, to enable them to perform as part of a CIM system.

Primary key (or tuple)

A key which uniquely identifies a record (or data grouping).

Product introduction

The activities associated with specification, design, analysis, process and resource planning and other functions required to bring a product to the production stage.

Proprietary

Belonging or under the control of a private organisation (e.g. AUTOCAD is a proprietary package, SNA is a proprietary protocol).

RDA

Remote Data Access.

Relation

A two-dimensional array of data elements (implemented as a table in a relational database).

Relational Database

A database made up of relations (as defined above). Its database management system has the capability to form different relations thus giving great flexibility in the usage of data.

Schema

A structured description of the information available in a database.

SME

Small and medium sized enterprises.

Table

A collection of data (in a relational database) suitable for quick reference, each item being uniquely identified either by a label or its relative position.

Turnkey

A turnkey system is one which is delivered and implemented by the supplier with little effort on the part of the user. It is largely "pre-designed" (at least at the component level) and the user sacrifices a close match to his own requirements in order to be able to be "up and running" quickly.

LIST OF FIGURES

LIST OF TABLES

SYNOPSIS

The introduction and adoption of contemporary computer aided manufacturing control systems (MCS) can help rationalise and improve the productivity of manufacturing related activities. Such activities include product design, process planning and production management with CAD, CAPP and CAPM. However, they tend to be domain specific and would generally have been designed as stand-alone systems where there is a serious lack of consideration for integration requirements with other manufacturing activities outside the area of immediate concern. As a result, "islands of computerisation" exist which exhibit deficiencies and constraints that inhibit or complicate subsequent interoperation among typical MCS components. As a result of these interoperability constraints, contemporary forms of MCS typically yield sub-optimal benefits and do not promote synergy on an enterprise-wide basis.

The move towards more integrated manufacturing systems, which requires advances in software interoperability, is becoming a strategic issue. Here the primary aim is to realise greater functional synergy between software components which span engineering, production and management activities and systems. Hence information of global interest needs to be shared across conventional functional boundaries between enterprise functions.

The main thrust of this research study is to derive a new generation of MCS in which software components can "functionally interact" and share common information through accessing distributed data repositories in an efficient, highly flexible and standardised manner. It addresses problems of information fragmentation and the lack of formalism, as well as issues relating to flexibly structuring interactions between threads of functionality embedded within the various components. The emphasis is on the :

- definition of generic information models which underpin the sharing of common data among production planning, product design, finite capacity scheduling and cell control systems.
- development of an effective framework to manage functional interaction between MCS components, thereby coordinating their combined activities.
- "soft" or flexible integration of the MCS activities over an integrating infrastructure in order to (i) help simplify typical integration problems found when using contemporary interconnection methods for applications integration; and (ii) enable their reconfiguration and incremental development.

In order to facilitate adaptability in response to changing needs, these systems must also be engineered to enable reconfigurability over their life cycle. Thus within the scope of this research study a new methodology and software toolset have been developed to formally structure and support implementation, run-time and change processes. The toolset combines the use of $IDEF_0$ (for activity based or functional modelling), $IDEF_{1X}$ (for entity-attribute relationship modelling), and EXPRESS (for information modelling).

This research includes a pragmatic but effective means of dealing with legacy[1] software, which often may be a vital source of readily available information which supports the operation of the manufacturing enterprise. The pragmatism and medium term relevance of the research study has promoted particular interest and collaboration from software manufacturers and industrial practitioners. Proof of concept studies have been carried out to implement and evaluate the developed mechanisms and software toolset.

SCOPE

The author's underlying philosophy, concepts, derived methodologies, and enabling mechanisms have been conceived to (i) facilitate software interoperability and (ii) overcome certain limitations and restrictions found when achieving applications integration in contemporary forms of MCS. Such issues have been considered in chapters one to five of this thesis. In chapter six the notion of a "model driven" methodology is introduced which has been derived to formally structure and support change management in interoperating forms of MCS. Here the concept of "enacting" functional and information models is examined to provide consistent support over the MCS life cycle. Chapter seven focuses on the implementation of a proof of concept MCS demonstration system to illustrate the application of the research methodology adopted and developed by the author to enable interoperation of MCS components.

Chapter 1 : The need for software interoperability and the issues which affect integration between software applications are discussed. Some of the major considerations are examined to develop pragmatic and viable interoperable systems. Also outlined are :

- the nature and scope of software interoperability;
- a requirement specification to effectively enable interoperation between MCS components.

[1] The term legacy (systems, components and software) is used to refer to a previously installed base of systems, components and software. Legacy elements will not normally conform to the methods and standards which are adopted in current generation solutions.

Chapter 2 : Inherent constraints and limitations which inhibit or complicate interoperation between contemporary forms of MCS are identified and examined. A literature review on contemporary approaches and methodologies available to tackle particular aspects of the interoperability problem(i.e. interconnection, system design and development, reference models and system behavioural issues) is conducted. Based on the review, some of the outstanding issues and "gaps" in knowledge are highlighted in terms of advancement goals for the author to achieve and address in order to effectively facilitate software interoperability.

Chapter 3 : The focus and perceived objectives of this research study are specified. The need for a reference information model, which can be widely applicable across the manufacturing continuum to promote interoperability between MCS functional activities, is highlighted. The components of an integrating infrastructure (IIS) and the important role of the CIM-BIOSYS IIS, which has been developed by the Manufacturing Systems Integration Research Institute at Loughborough University of Technology and used in the author's research study to enable interconnection between MCS components, are described. Also provided is an overview of the overall research methodology conceived by the author as part of this doctoral study to enable software interoperability.

Chapter 4 : An information architecture which can support software interoperability is discussed. It includes the specification of reference models, which describe information entities of common interest shared between production planning, product design, process planning, finite capacity scheduling and cell control systems. Included is a case study carried out in collaboration with the University of Bradford Management Centre to demonstrate the application of the generic reference models identified and described by the author. The design criteria for the system data repository are specified. The mechanisms (such as the database 'driver') developed to enable open information access in a consistent and reliable manner from the system data repository are also described.

Chapter 5 : The emphasis in this chapter is on system behavioural issues. The methodology adopted, mechanisms developed and system management tools required to formally structure and facilitate functional interaction (among distributed interoperating MCS components during run-time) are discussed. The MCS Functional Interaction Management Module (FIMM), which has been developed as part of this research study to enable effective functional interaction, is also explained. The functional capabilities and level of effectiveness of the MCS FIMM in facilitating interaction is compared against a contemporary commercial software application, namely Systems Integration Manager, which is designed to tie together a disparate set of MCS software applications into a coherent system.

Chapter 6 : A "model driven" methodology conceived by the author in this research study to provide life cycle support of integrated manufacturing systems is highlighted. The software toolset developed, which is coupled closely with IDEF_0, IDEF_{1X}, and EXPRESS modelling tools, to formalise and support implementation, run-time and change processes is described. The exploitation and enactment of function and information models by the software toolset to structure downstream life cycle processes is also discussed.

Chapter 7 : In this chapter a proof of concept MCS implementation study has been carried out in order to illustrate the application and ascertain the level of effectiveness of the methodology derived in this research study, which seeks to enable interoperability among MCS components.

Chapter 8 : A summary of the research work carried out is provided. In order to enhance the software interoperability methodology described in this thesis, future work is recommended to address some existing deficiencies. Expected future major areas of interoperability development are also highlighted.

Chapter 1

Contemporary forms of Manufacturing Control Systems and a perspective on Software Interoperability

> *"He who does not think far ahead, is certain to meet troubles close at hand."*
> Confucius, Born 551 BC

1.1 Limitations of Contemporary forms of MCS

Advances in technology have led to the widespread use of information technology in manufacturing but have mainly generated islands of computerisation [Hars 1990]. This phenomenon can in part be attributed to the dominant influence of Taylorism [Pugh and Hickson 1989, Taylor 1947], which places emphasis on specialisation and distinct division of responsibility. Traditionally this has led to a compartmentalization of business, engineering and production activities, all of which are required to support the product life cycle. Consequently, there has been a proliferation of *computer-aided software applications where each has been designed with Tayloristic principles in mind, this to address some focused aspect of the manufacturing domain* [Scheer 1988].

Contemporary software applications are used to facilitate product introduction and planning processes. They also enable the control of manufacturing activities and processes on the shop floor. They are implemented in a variety of forms and serve different purposes, many of which are typically classified under the general headings of PPC, CAD, CAM, CAE, CAPP, CAQ, DNC, SFC, etc.) [Rembold *et al.* 1993, Scheer 1991]. Thus typical components of contemporary MCS[1] (Manufacturing Control Systems) are implemented in the form of stand-alone software packages, each focused on enabling localised efficiency and productivity improvements with respect to different aspects of a manufacturing enterprise. Computer-aided software applications of this type are conceived, implemented and supplied from various sources, this leading to significant heterogeneity in terms of the computer hardware, systems software, systems management and communications systems they employ [ITAP 1990, Weston *et al.* 1988]. This is accentuated by the fact that they are conceived and implemented in the *absence of an overall company wide strategic plan* [Waterlow and Monniott 1986] *of how MCS should be implemented and how the integration requirements of other software*

[1] MCS - Specialised applications which exhibit discrete functionality for the rationalisation, improvement, control, and execution of activities in support of part manufacture will be viewed as typical components of MCS.

applications, outside the area of immediate concern, can be met. Driven by myopic departmental considerations, end users freely mix and match hardware and software requirements to best suit their needs, from the plethora of systems that are currently available.

As a result, contemporary MCS solutions offer little or no interworking between constituent software applications, particularly those associated with different functional areas of the enterprise, and do not promote synergy on an enterprise-wide basis [Moerman 1991]. This results in specialised departmentally-determined data organisation and contributes to data hoarding by *individual departments and applications, each with its own restricted and incomplete view of enterprise goals* [DTI 1993]. Companies are often in a dilemma when they need to :-

- accrue and consolidate fragments of information of common concern which are distributed and also duplicated among the various MCS applications; and
- facilitate information sharing and transfer between islands of computerisation.

This can be attributed mainly to the insular nature of contemporary MCS solutions where their interconnections (to facilitate transfer and sharing of information) are severely constrained by their proprietary nature and the level of heterogeneity exhibited by them [Singh and Weston 1993, Weston *et al.* 1988]. Hence *there is much reliance on users to establish informal links to (i) convey messages and information, and (ii) co-ordinate transactions between specialised departments and applications,* as illustrated in Figure 1-1. Consequently, significant delays and errors in transactions normally occur, thereby promoting opportunities for misunderstandings and conflicts [Lars 1990]. This undoubtedly hampers the productivity and operational efficiency of the enterprise concerned.

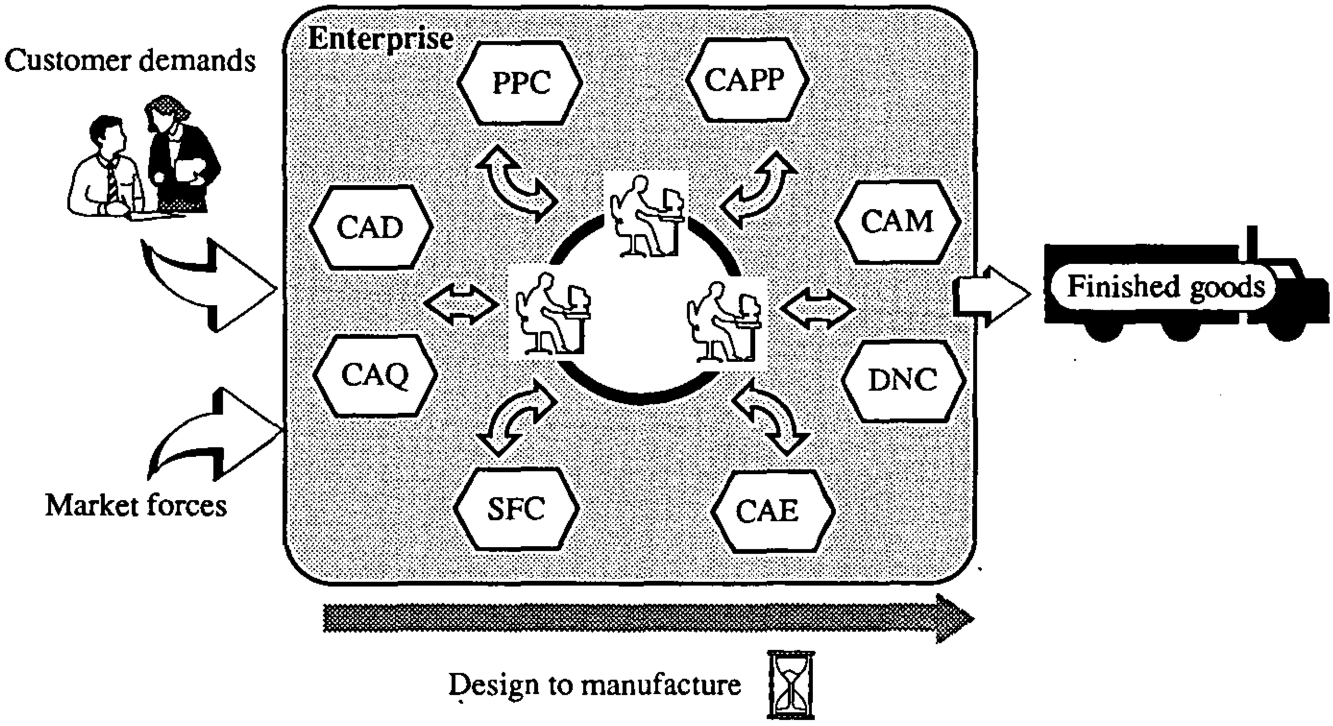


Figure 1-1 : Insularity of contemporary MCS solutions

1.2 The Need for Software Interoperability

With a growing demand for consumer-oriented flexibility, there is increasing pressure on companies to improve their responsiveness and to achieve better ("more informed") decision-making through effective dissemination and sharing of information and knowledge [Pheasey 1992, Weinberg 1989]. These needs have been identified and are strongly reflected in the following emerging trends [Fry and Baker 1993, Peters 1989] :

- **Inter- and intra-organisation integration** to realise greater functional synergy where conventional boundaries between enterprise functions, which span engineering, production and management activities and systems, are traversed to allow information of global interest to be shared within the organisation and externally, such as with vendors and suppliers.
- **Time-based competition** where the drive is towards achieving shorter design to manufacture lead time in order to be highly responsive to customers' demands and changes in other market forces so as to enable exploitation of market potential and capitalisation of existing business opportunities.
- **Customisation** of manufacturing activities and systems where they need to be configured and geared specially to cater for the individual requirements and idiosyncrasies of the company concerned.

Therefore, it is no longer reasonable to expect a single software application to fulfil its purpose without support or reference to data and events which are handled by other closely related application systems. This would require the linking up of the various islands of computerisation, for example :

- **Product Design**

  Companies are under constant pressure to (i) deliver better quality products, and (ii) dramatically reduce design-to-manufacture lead times so as to capitalise on market opportunities by being "first to market" [DTI 1993]. Therefore, ***not only must there be prompt responses to design requirements and changes but there must also be the use of decision support systems based on information gathered from various parts of an enterprise*** [DTI 1993, Schnur 1987]. For example, to aid product design, product specifications should consider the availability of manufacturing resources [Singh 1991, Lars 1990].

- **Process Planning**

  This serves as a technological bridge between engineering and manufacturing and provides a blueprint for part manufacture [Ssemakula 1987, Logan 1986, Chang 1985]. It defines the sequence of operations required to manufacture a product and selects the manufacturing resources (including machines, toolings, fixtures) to carry out material processing. According to a recent report [Halevi and Weil 1992] ***process planning systems should be able to communicate with other company functions, especially production planning and product design, in order to achieve significant benefits in co-ordinating manufacturing activities.***

- **Cell Control**

  Shop and cell control systems have responsibilities for a segment of the shop floor and are required to despatch planned orders (which are generated from production planning systems), coordinate, control and monitor the operation of the components of a shop or cell [Bauer 1991]. They will also be responsible for shop floor data acquisition, thereby enabling production status feedback to the production planning system [Williams and Rogers 1991].

Thus it is necessary to coordinate and control the interoperation of various types of software component; this involves a need to provide and control access to commonly used engineering, manufacturing and management information (see Figure 1-2). For example, bill of materials (BOM) and process planning information are common information entities viewed from different perspectives by CAD/CAM [CIM Strategies 1991, Lang-Lendroff and Unterburg 1989, Bohse and Harhalakis 1987] and CAPP [Ssemakula 1987, Logan 1986].

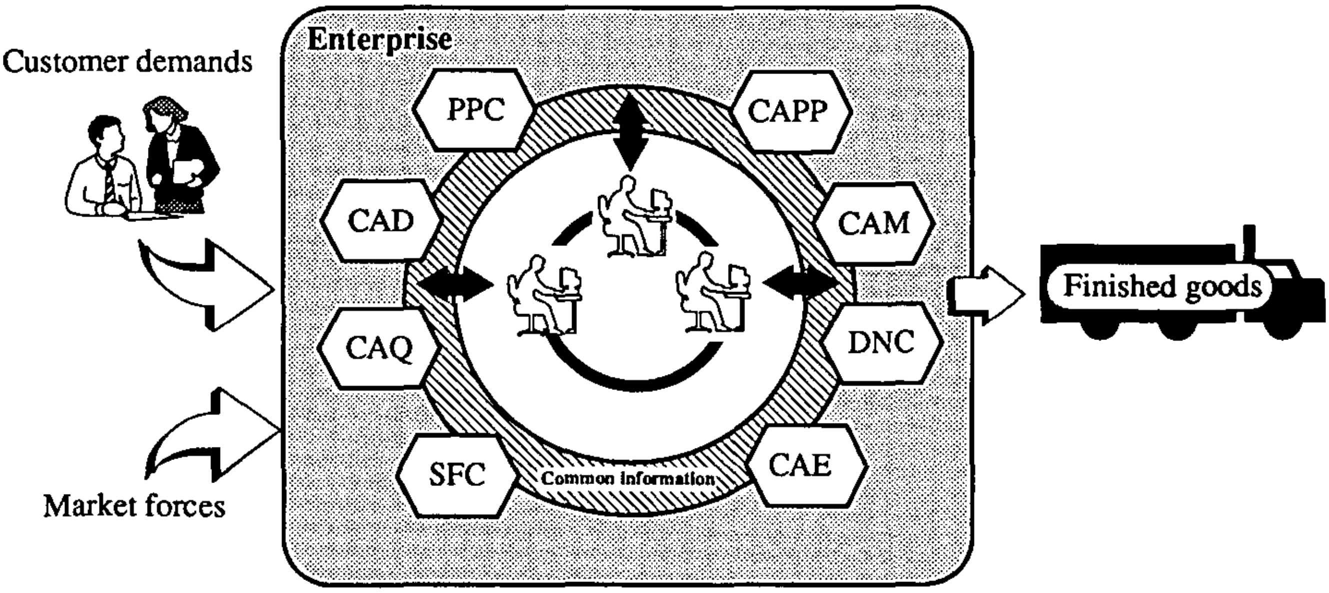


Figure 1-2 : Interoperation among MCS solutions

As illustrated in Figure 1-2, the users, on the other hand, would now require

(i) accessibility and a coherent as well as up-to-date view of information of common concern, this to support their decision-making in a more timely and accurate manner; and

(ii) a global perspective of their work domain, so as to have a better appreciation and understanding of its impact on and close association with other activities in the enterprise.

This would undoubtedly help to avoid errors and delays in and between transactions as well as alleviate potential conflicts and contentious situations.

However, the following should be considered and addressed before any solution derived can be deemed as practically acceptable :

- dealing with legacy software, this to enable interoperability among the existing installed base of MCS components.
- ensuring generic applicability of interoperable solutions across the manufacturing continuum which typically range from 'Project Manufacturing' to 'Repetitive Manufacturing' [Dinitz 1990].

1.2.1 Coping with Legacy Software

Contemporary MCS are normally designed and implemented to address a set of problems prevalent in current company situations. Once installed, inevitably any solution is at a risk of becoming obsolete if it is incapable of coping with situations other than those for which it has been designed. Hence MCS solutions are required to (i) adapt to changes in business needs, and (ii) conform to methods and standards adopted in current generation solutions. When obsolescence occurs, one of the following actions will normally be required :

(a) modification or enhancement in functionality so as to upgrade and make them functionally effective again.

(b) to retrieve its existing information which can be a vital resource in support of the operation of the manufacturing enterprise.

(c) discarding them entirely and replacing them by a viable alternative when there is neither any chance nor need for (a) and (b).

With a previously installed base of a software, which is referred to as legacy (or "as is") software, the possibility of performing (a) may be remote. Often this is due to a lack of proprietary knowledge, support and expertise to carry out necessary changes (such as amendments to the required source programs), thereby making the task arduous indeed [Singh and Weston 1993]. Furthermore, it is uncommon for software manufacturers to reveal and release information pertaining to the source programs of their developed applications. Understandably, this is to protect their vested commercial interest and rights to intellectual property embodied in their products.

Hence in seeking to facilitate information sharing, generally speaking (b) seems to be a more workable and pragmatic means of dealing with legacy software than (a). However, in order to achieve an acceptable level of interoperability among MCS components, often the following prerequisites are essential :

- that the information source (which forms part of the legacy element) can be independently accessed; and
- that the information architecture and schema used by the legacy component are clearly understood in terms of their structure and composition [Hodgson and Weston 1993].

1.2.2 Manufacturing Continuum Consideration

Typically, the production planning methods and systems employed today are likely to have been chosen after having considered particular characteristics of the manufacturing environment concerned, i.e. the type of products, production volume, demand fluctuations, manufacturing technology, markets involved, etc. [Zäpfel and Missbauer 1993]. Indeed there is a spectrum of discrete parts manufacturing which can be viewed as ranging from 'Project Manufacturing' to 'Repetitive Manufacturing' [Dinitz 1990] (see Figure 1-3). Not surprisingly therefore *there is considerable diversity and lack of uniformity among production planning methods and systems.*

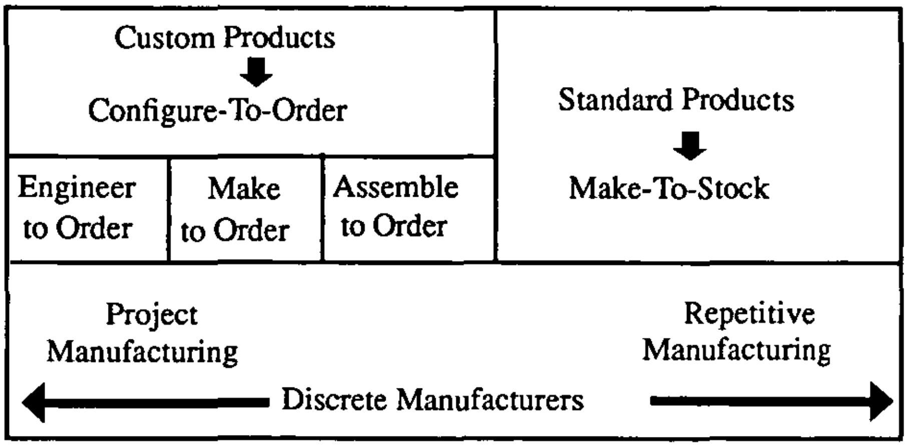


Figure 1-3 : Continuum of discrete part manufacturing environments [Dinitz 1990]

Clearly, the environment required for each class of manufacture (be it 'engineer-to-order', 'make-to-stock', etc.) will require its own distinct set of operating characteristics [Goyal *et al.* 1993], as illustrated in Table 1-1.

	Manufacturing Environment	
	Engineer-To-Order	**Make-To-Stock**
Production Complexity	High	Low/Medium
Capacity/Material Driven	Material	Capacity/Material
WIP Value	High	Low/Medium
Push/Pull	Pull	Push
Schedule vs Orders	Orders	Schedule
Forecast Stability	Low	Medium/High
Direct Issue vs Backflush	Direct	Backflush
Shop Floor Organisation	Work Centre	Line/Cell
Manufacturing operations Staff	High	Low/Medium
Labor Content	High	Low/Medium
Overhead Basis	Labor $, Labor Hrs, Machine Hrs	Labor $, Labor Hrs, Machine Hrs
Performance Goals	Operation Efficiency	Schedule Attainment

Table 1-1 : Characteristics of discrete manufacturers [Goyal *et al.* 1993]

Hence suitable manufacturing methods have to be adopted to satisfy and address the specific needs of each class of manufacture. They govern the formulation and determine the nature and extent of MCS functionality, particularly with regard to production planning and control. Common manufacturing methods adopted include MRP II, JIT, OPT or possibly a hybrid of them [Zäpfel and Missbauer 1993, Larsen and Alting 1993, Struedel and Desruelle 1992, Higgins *et al.* 1991, Jones and Roberts 1990, Goldratt 1988, Wight 1984] where

- MRP II operates on forecasts, time-phased resource planning and fixed lead times. It is suitable for discrete part manufacture of standard products for the 'make-to-stock' environment. However, it does not include functionality to support short term scheduling of orders under the constraints of the real availability of resources. This can make the use of

MRP II inflexible, capacity insensitive and not responsive enough to changes and short term demands imposed by, for example: the 'configure to order' environment, which is characterised by small batch quantities and large product varieties.

- OPT (Optimised Production Technology) is based on the insight that the flow of materials and goods, consequently affecting the performance of the manufacturing system, is determined by properties of its bottlenecks, such as limited capacity, demand and availability of raw materials.
- JIT (Just-in-Time) methods are not only used to reduce inventories but also for continual improvement of the production process. This approach requires a reorganisation of the logistic chain to provide sufficient flexibility and reliability to closely match resources and capability to customer demand.

The strength of MRP II lies in its mid- to long-term global planning whereas the strengths of JIT and OPT are in the short term execution of planned needs. Each has proved to be successful in certain production environments and each has demonstrated disadvantages under certain conditions. It is beyond the scope of this research to discuss further these paradigms and as each of them is sufficiently complex, no attempt will be made here to elaborate upon the various philosophies which they embody. Instead, the reader is directed to one of the references given.

However, bearing in mind the following realities, *there is a need to scrutinize the (information and functional) requirements of the various manufacturing methods so as to identify amongst them (i) essential information of prime concern, and (ii) the existence of common threads of functionality*:

- In practice companies do not necessarily stop at the boundaries of one manufacturing method but cross and mix ideas to extract what makes sense for that particular company [Van Donselaar 1992, De Vaan 1992, Ptak 1991, St. Charles 1987, Luscombe].
- Any choice of manufacturing method depends upon various drivers (for example, the nature of the management style, organisation, information technology and manufacturing technology), each of which will inevitably change and evolve, as illustrated in Figure 1-4.

Quite importantly, such generalisation would *alleviate any bias towards specific manufacturing methods, thereby effectively breaking free from the restrictive bounds of the manufacturing methods adopted.* In addition, commonality in information requirement and functions would indeed help overcome the considerable diversity among production planning methods and systems, currently existing across the manufacturing continuum, which serve to inhibit the achievement of interoperable solutions.

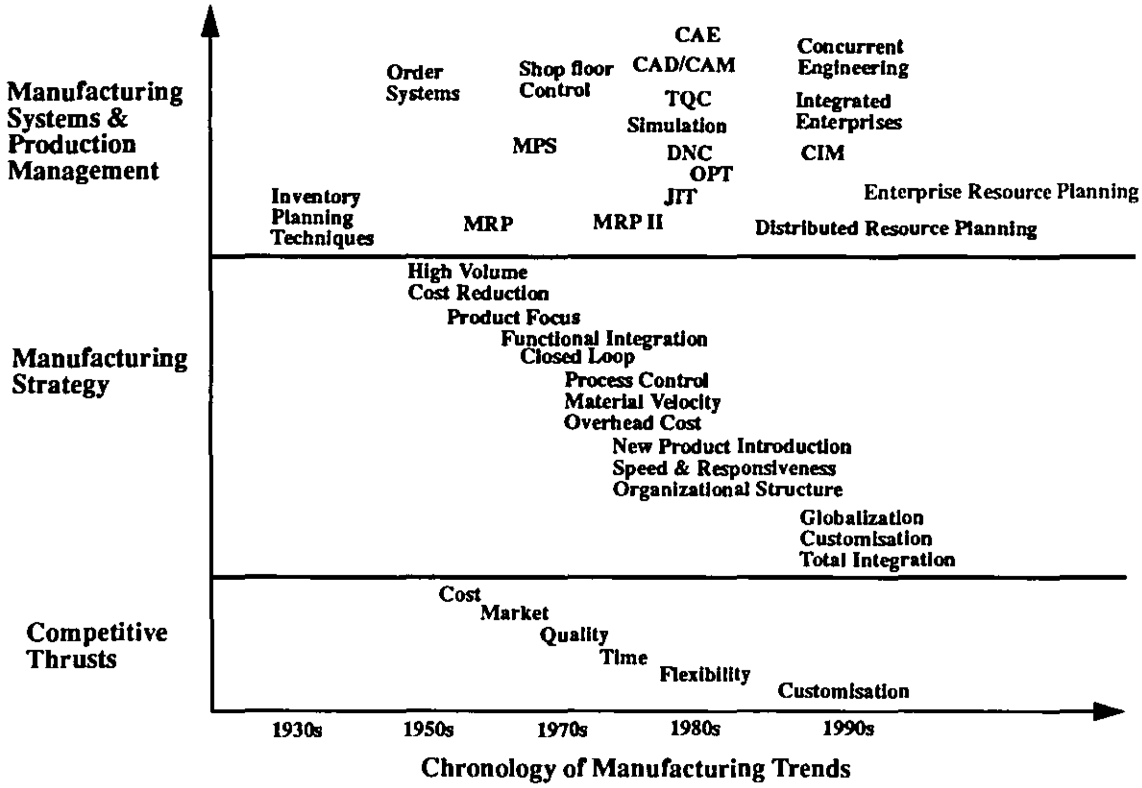


Figure 1-4 : Evolution of production management and manufacturing systems

1.3 Applications Integration and Software Interoperability

As illustrated in Figure 1-5, the AMICE CIM-OSA Consortium classified integration in the manufacturing enterprise within the following three levels [ESPRIT 1989, CIM-OSA 1989] :

- **Business Integration**

  At this level, enterprise goals and strategic business issues are considered.

- **Application Integration**

  Integration at this level is defined as concerning interoperation between applications, this to facilitate data sharing and information exchange.

- **Physical Integration**

  According to CIM-OSA, physical integration is mainly concerned with data and inter-process communication issues. It expects this level of integration to be provided by current information technology concepts and standards (such as the ISO/OSI seven-layer reference model [DATAPRO 1992], as illustrated in Figure 1-6). CIM-OSA will use the relevant services as defined.

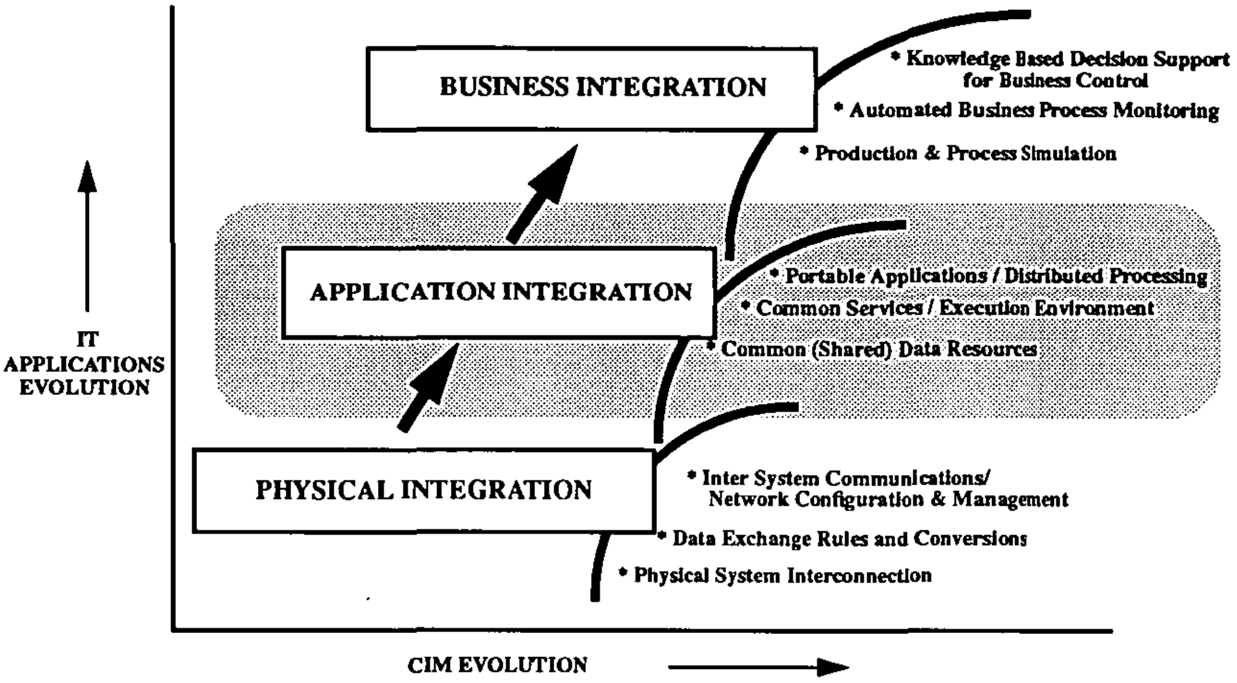

BUSINESS INTEGRATION
APPLICATION INTEGRATION
PHYSICAL INTEGRATION
IT APPLICATIONS EVOLUTION
CIM EVOLUTION
* Knowledge Based Decision Support for Business Control
* Automated Business Process Monitoring
* Production & Process Simulation
* Portable Applications / Distributed Processing
* Common Services / Execution Environment
* Common (Shared) Data Resources
* Inter System Communications/ Network Configuration & Management
* Data Exchange Rules and Conversions
* Physical System Interconnection

Figure 1-5 : Integration levels in the manufacturing enterprise [CIM-OSA 1989]

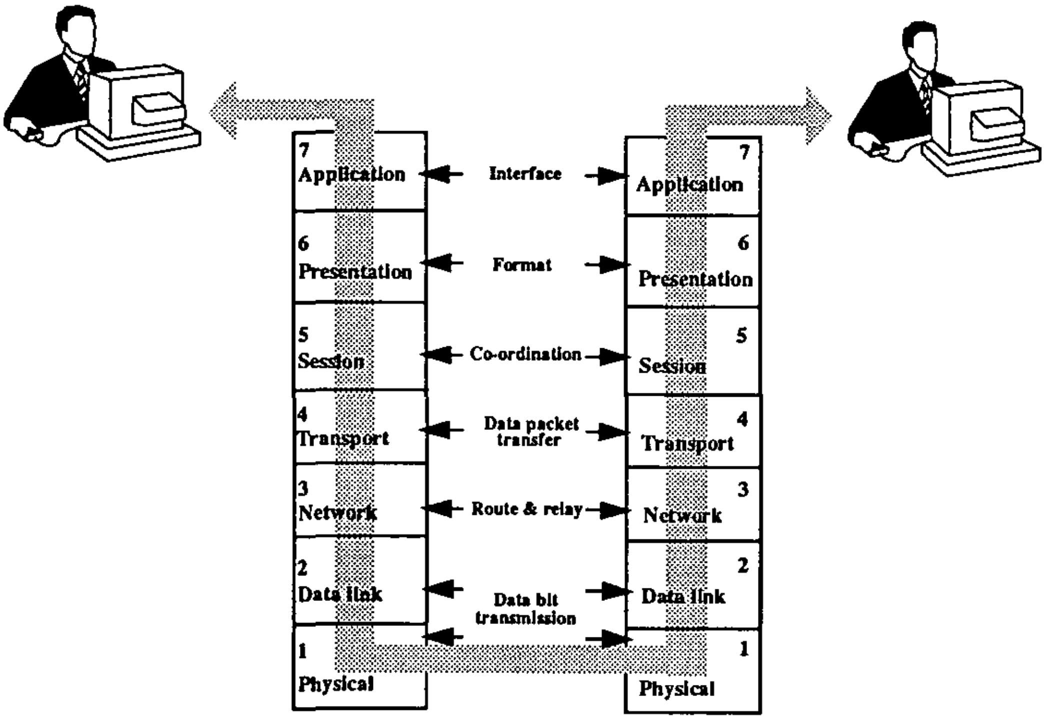


Figure 1-6 : Illustration of ISO/OSI seven-layer reference model

Within the context of computer integrated manufacturing (CIM), the term "software interoperability" has been used to imply the ability of separate software applications to functionally interact so as to meet collective goals [Pheasey 1992, Scheer 1991, CIM-OSA 1989]. Bearing in mind the CIM-OSA classifications, therefore software interoperability is aimed at the application integration level of enterprise wide integration.

Software interoperability is widely conceived as requiring data integration as well as functional integration [Singh and Weston 1994a, Anscombe 1992, Hars 1990, Scheer 1989, Solberg 1989, Fritsch 1989, Weinberg 1989, DTI 1987] with the resultant effect of linking and synchronising the behaviour of processes in different subsystems of an enterprise. *The underlying interaction processes will involve an exchange of messages and the sharing of information of common interest between a group of software applications so that the applications behave (both individually and collectively) in an effective manner whilst realising system-wide goals.*

Drucker, in his assessment of the factory of the future, expresses the importance of such a need as being :

> " *The factory of the future will be an information network. Sectors and departments will have to think through what information they owe to whom and what information they need from whom. A good deal of this information will flow sideways and across departmental lines, not upstairs as with traditional plant...*" [Drucker 1991]

Over the last decade significant progress has been made towards improving the 'hardware portability' of MCS software building blocks where software vendors have sought to adopt the use of de jure and de facto computer networks, operating systems, databases, fourth generation languages, and graphical user interface standards [Evans *et al.* 1993, DATAPRO 1992]; as illustrated in Figure 1-7.

This trend towards hardware portability has enabled the functionality implemented by any particular piece of application software to be separated from (i) the computer hardware on which it is run and (ii) the data on which it operates. The result is that certain problems associated with installing, using, and changing individual MCS software building blocks can be alleviated [Evans *et al.* 1993, AMR 1991]. However, in seeking significant enhancement in the level of interoperability achieved between chosen MCS software building blocks (where information exchange is a key requirement) *common function and information models, which encapsulate key general attributions of various forms of MCS, are also required* [Hodgson and Weston 1993, ISO 1991, Barkmeyer 1989, Weber and Moodie 1989]. Currently, there is an absence of such models.

Requirements	Standards		Functions
	De Jure	De facto	
Windowing	X-Window System OSF/Motif		Consistent User Interface
Graphics	GKS-3D PHIGS IGES		
Operating System Interface	(IEEE POSIX 1.003.1, Emerging 1.003.n)		Access to System Services
Database Definition/Access	SQL, Emerging SQL Access & Remote Data Access (RDA)		Information & Resource Sharing
Enterprise Repository	Emerging IRDS, ATIS		
File Sharing	Network File System (NFS) Emerging OSF DCE AFS	TCP/IP	
Mail	X.400		Enterprise Communications
EDI	ANSI X.12, EDIFACT	EDIF	
RPC	Emerging OSF DCE		
Plant Floor Communications	MMS	RS-232	

Figure 1-7 : The standards continuum and a categorisation of their purpose

Thus *as an initial step, there is an important need to define the nature and form of a suitable information model which can be used and advanced to a point where it can serve as a generalised reference model, thereby facilitating data exchange between MCS components* [ISO 1991, Barkmeyer 1989]. The benefits of using this reference model need to be quantified and widely published. Subsequently it would be necessary for the reference model to be adopted by vendors and users of MCS where it is likely that significant additional benefits would accrue as its applicability and adoption are widened [Singh and Weston 1994b]. Such a model will need to describe information of common interest to the components of a typical MCS, thus effectively serving as a precursor to their interworking. It should reference shared information and possibly consolidate them in a data repository so as to enable data sharing within the enterprise. Potentially using such an approach, boundaries between enterprise-wide functions can to some extent be overcome, a key requirement as advocated by Anscombe [Anscombe 1992].

1.4 Requirement Specification for Software Interoperability

We can conclude that **ideally software interoperability should imply an uninhibited functional interaction and intercommunication amongst CIM components through a free exchange of shared information.** This also implies a need to standardise the interfaces between the software components of CIM systems, especially concerning the control of information exchange between the components. Therefore, it is clear that the concept of software interoperability extends beyond that of software portability, i.e. interoperation is required over different hardware platforms, operating systems and information access and storage systems.

Hence in summary, the following requirements need to be satisfied to enable software interoperability in an effective manner which overcomes (i) limitations inherent in contemporary MCS components and solutions; and (ii) associated and inherited difficulties and problems involved in achieving their interoperation :

* ***Information sharing requirements***

- Generic reference models which describe information of common concern to various components of MCS.
- An information architecture which establishes structure and uniformity whilst enabling sharing and transfer of information between MCS components.

* ***Interconnection facilities***

- An integrating infrastructure which simplifies and structures interconnection by (i) separating integration and application issues; (ii) providing inter-process communication services; and (iii) mapping of distributed processes (embodied in MCS components) on the physical resources contained within a target manufacturing system.

* ***User interface capability***

- Ability for users to access MCS related functions in a manner which provides a global perspective and intra-organisation support for their tasks.

* ***Control of system behaviour***

- Capability for controlling and co-ordinating the sequence of (run-time) activities carried out by an MCS, this based on their functional dependencies, information needs and availability.

* ***System design and development capability***

- Provision for a more formal and structured approach to
  - engineering MCS solutions; and
  - supporting them over their useful life.

This is to facilitate ease of development and change management, in view of the need for next generation forms of MCS to be adaptable and responsive.

Chapter 2

Current "State-of -the-Art" Scenario

2.1 Introduction

There are various methods currently used which attempt to resolve and overcome (i) certain limitations inherent in contemporary MCS components and solutions, and (ii) difficulties and problems associated with achieving a degree of software interoperability. In this chapter, the nature and status of these methods is discussed. The discussion is structured with reference to certain aspects of the requirements specification identified in Section 1.4, namely with respect to commonly used ways of providing :

- ***Interconnection facilities***
- ***Information reference models***
- ***System design and development capabilities***
- ***Means of controlling system behaviour***

2.2 Interconnection Facilities

In this context interconnection can be viewed as establishing electronic data interchange between the various islands of computerisation [Rembold *et al.* 1993, DATAPRO 1992]. Here components of an MCS are interconnected to provide a low level data inter-communication and information transfer facility for the software applications (or components) which form the MCS. Three broad classes of approach can be identified which will herein be referred to as :

- **"Pair-wise" integration**
- **Integrated information systems**
- **Integrating infrastructure**

2.2.1 "Pair-wise" Integration

As explained by Rui in his PhD thesis [Rui 1989], in industry a "pair-wise" integration approach is frequently adopted to interconnect the components of an MCS, thereby enabling information transfer between software applications (or components) of such systems. This approach can be characterised as follows :

- **Requires the development of bespoke interfaces**, as illustrated in Figure 2-1. These interfaces will normally need to be custom designed (each at a relatively high cost) to realise a level of integration between interoperating pairs of application software. The *complexity of such systems will grow substantially (theoretically in a square law fashion) as the number of interconnected applications grows* [Weston 1993]. This implies that for a set of n different systems which need to be linked, n(n-1) different interfaces may need to be developed; what is worse is that potentially 2(n-1) interfaces need to be adapted whenever a single application system is changed.

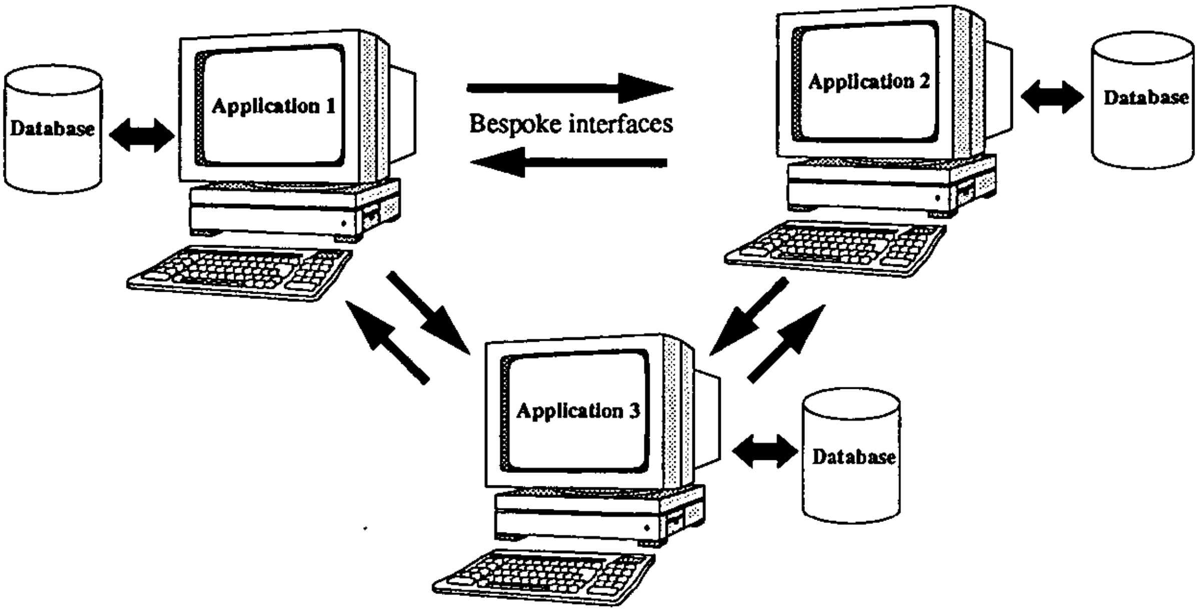


Figure 2-1 : Interconnection between MCS applications through "pair-wise" integration

Typically this approach results in the incorporation of knowledge (concerning the need to interoperate) into individual application software and its associated 'drivers' [Rui 1989]. This will include knowledge of other application software, data sources, data access mechanisms, communication protocols, communication channels, data formats, and data structures [Kaul *et al.* 1989]. *This results in inflexible, application-specific and rigid solutions which can be classified as "hard" integration"* [Rui 1989]. Such solutions will not be easily supported (in terms of available technical expertise) and the understandable reluctance of vendors to release detailed product specifications (as this may embody knowledge which provides them with a competitive advantage) will often lead to sub-optimal interworking between components. Furthermore, the cost (in terms of resources and time) of subsequent modification may be so great as to render the solution obsolete as soon as requirements change significantly.

- **Utilisation of import and export filters.** For some software packages, export filters are provided as a built-in utility to enable the user to have independent access to its proprietary data [Preece 1993, DATAPRO 1992]. The filters assume responsibility for pre- and post-processing of data which is specifically selected by the software manufacturer to be made available to the user. Here some restrictions are necessary in order to maintain data security and integrity through controlled access. The data will be automatically converted, via the filters (see Figure 2-2), to conform with a required database or file format supported by the software package. Generally the file formats adopted here will either be :

  - a compatible format to enable direct data transfer between software packages; or
  - a neutral format, such as in the form of a flat file, to allow intermediate data transfer (in the absence of any compatible database or file format) between software packages. In this case, further overhead processing will be required to retrieve, manipulate and store the relevant data.

  As explained by Lim [Lim 1992], data conversion and data transfer is normally performed in a batch mode processing.

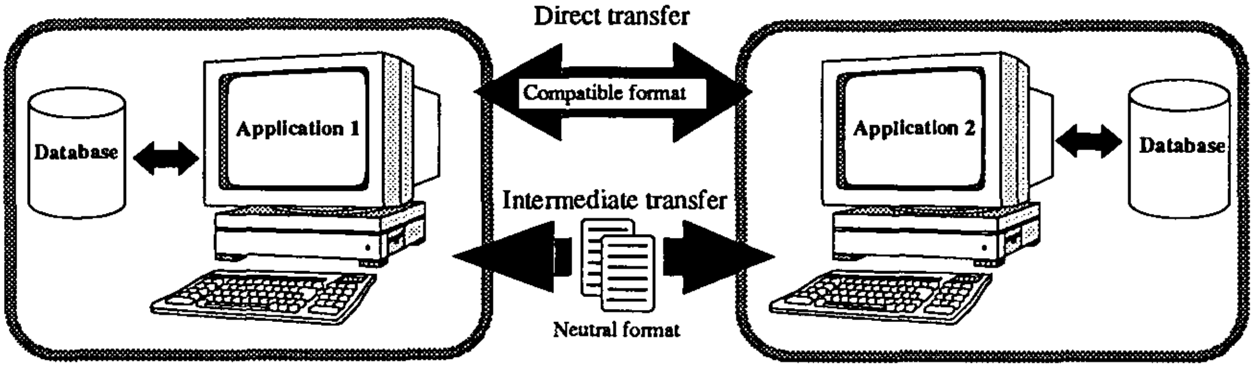


Figure 2-2 : "Pair-wise" integration via import/export filters

The main limitations of using filters are (i) that they provide only restricted access to selected data, and (ii) the details of data format and structure can be lost in the transfer process [Preece 1993].

With such contemporary "pair-wise" methods of interconnection, each software application manages its own data and this can result in significant access times and data transfer times, this as a result of the inherent mechanisms used to retrieve and make data available to other applications [DATAPRO 1992, Scheer 1991]. As a result of such delays there is no guarantee that data will be sufficiently up-to-date to support other dependent applications.

2.2.2 Integrated Information Systems

There is a growing emphasis on the development of integrated information systems based on a data integration approach. This approach can help prevent cumbersome infomation transfer (as illustrated in Figure 2-1) and can also reduce delays in information transfer times [Muhlemann *et al.* 1991, DTI 1989]. ***The underlying principle of integrated information systems is that they consolidate information into a data repository (or common pool), this by seeking to closely map common data to be exchanged and shared between applications into that pool*** [Jeng and Chao 1992, Martin 1988]. Generally speaking, each application is required to support a capability to export and import schema to this data repository for which a global schema of common data models is defined (refer to Figure 2-3 for illustration). Potentially this approach should enable information of common concern (which typically will accrue at one stage in the production chain) to be included in the data repository, thereby making it accessible to application software used at other stages in the chain.

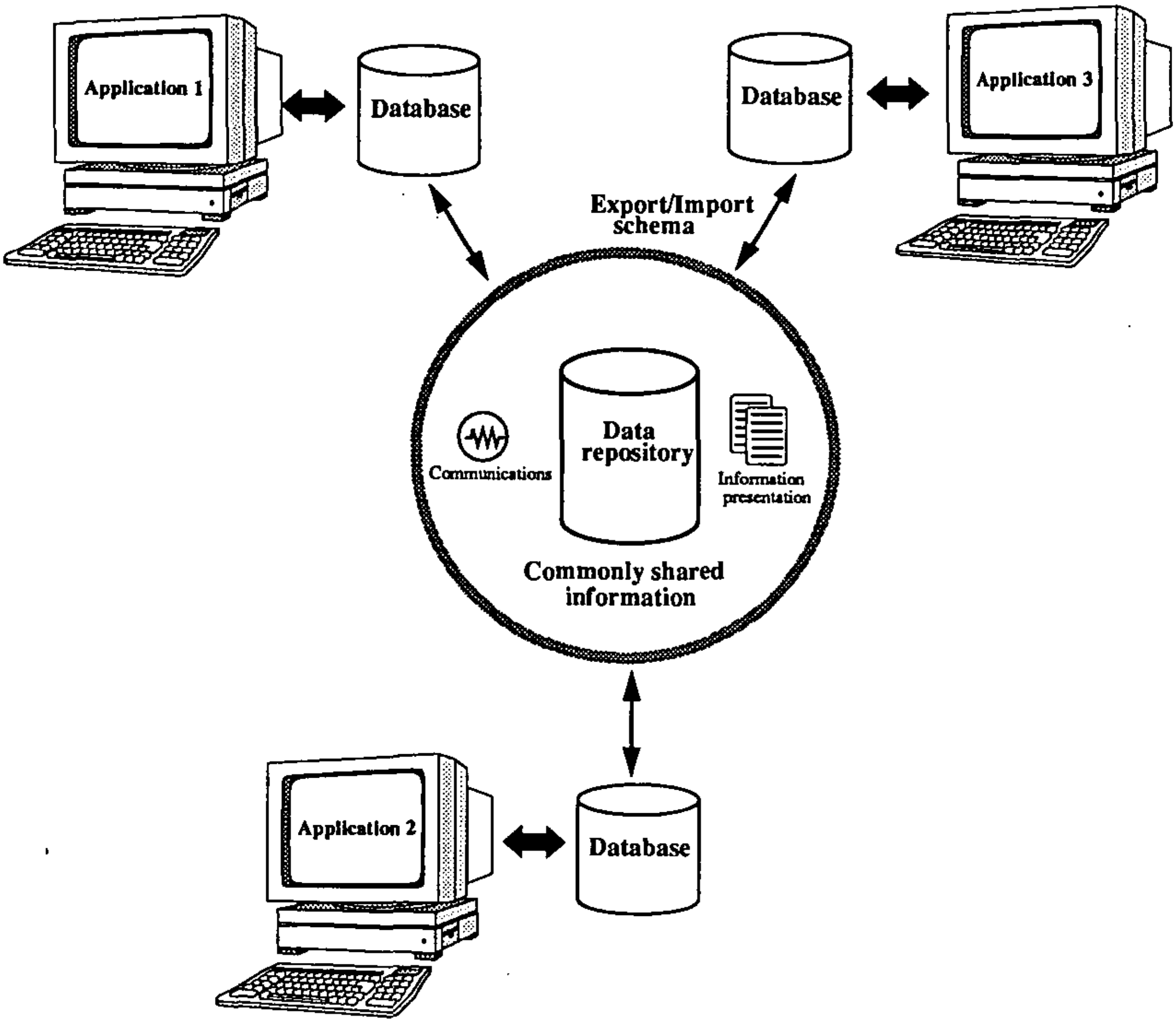


Figure 2-3 : Interconnection between MCS applications through integrated database

This approach has been realised industrially (at least to some extent) in recent years and has led to a degree of rationalization within divisions of some enterprises. For example, rationalisation spanning accounts, production planning and order handling have led to a reduction in administrative order handling times from 3 weeks to 3 days [Scheer 1991, Hars 1990].

However, as described below major practical problems remain with regard to the development, enhancement and maintenance of integrated manufacturing systems which impede the wide-spread adoption of integrated information systems and may generally restrict interoperation among MCS functions [Singh and Weston 1993, Kochhar *et al.* 1987]. Many of these problems arise from :

- *The tight-coupling that normally exists between MCS functions and their associated information.* This (i) makes information access difficult or even impossible; and (ii) leads to unintended propagation of the effect of changes made, i.e. change to individual applications (comprising software processes and their associated systems) can have significant effect on the operation of other applications [Weston *et al.* 1988].

- *A lack of adherence to standard architectural models of functionality and information.* Rather, contemporary software components of MCS are designed using proprietary models of function and information which are determined by the manufacturer [Singh and Weston 1994b, Fritsch 1989, Solberg 1989]. The main disadvantage is that information of common concern to software applications is often duplicated, translated, and re-interpreted by different software applications. This gives rise to problems of data integrity and consistency as well as significant database management problems [Lim 1992]. Furthermore, semantic integrity has to be ensured and maintained between valid combinations of data items fragmented across various databases [Kaul *et al.* 1989].

- *Heterogeneity in database systems, particularly with regards to their logical data model.* As elaborated below, there are the following three logical data models most commonly supported by database management systems [Beeri 1993, Wilkinson and Winterflood 1987, Date 1986, Martin 1980] :

  (i) Hierarchical model

  Data is represented in a hierarchical or tree structure. Tree structures provide a natural way of modelling truly hierarchical real world relationships where one-to-many segment types can be defined to represent successive levels in a tree structure in order to relate entities to one another.

(ii) Network model

In the network model data is represented in a network (or plex) structure where any node can be connected to any other node represented in the structure. Network structures offer a greater scope to represent data relationships than hierarchical structures, albeit at the expense of simplicity (at least with respect to physical storage structure).

(iii) Relational model

In a relational model entities, relationships and attributes are represented in the form of two-dimensional tables known as relations. Records are assimilated to the rows of the table and each set of attributes forms a column.

(Please refer to Appendix I for further details on the various types of logical data models). Thus there are major difficulties involved in attempting to interconnect heterogeneous database systems. As a result of non-uniformity in their database management systems and physical storage structures, there are serious concurrency problems related to transactions and controlled data access. In addition, the following must also be reconciled :

- differences in database schema;
- semantic differences among data items.

There is considerable academic and industrial interest in integrating heterogeneous distributed database systems, with extremely large numbers of publications in the area [Breitbart *et al.* 1993, Bright *et al.* 1992, Thompson *et al.* 1990, Motro 1987, Batini *et al.* 1986] indicated in the literature. The reader is directed to one of the references given for further appreciation of the difficulties involved.

2.2.3 Integrating Infrastructures

An increasing number of CIM tools [Gould 1992, AMR 1991, CIM Strategies 1990, Metz 1990] which are appearing on the market claim to allow applications to "functionally interact" (see Table 2-1 for summary). The main purpose of these integrating infrastructures (which will be referred to by the acronym IIS) is to structure, service, and where possible simplify interconnection between the component elements of software systems.

As illustrated in Figure 2-4, an IIS can be charged with resolving differences in a physical system relating to heterogeneity, distribution and data fragmentation [Weston 1993]. *It can assume responsibility for maintaining a knowledge of integration details* (such as the networks used, the hardware and operating systems that software components are run on, the location of an information fragment, etc.) so that software components (such as MCS components)

Vendor	Product	Remarks
IBM	PlantWorks/DAE DAE - Distributed Automation Edition	* Need compliance to IBM's SAA based strategy of a CIM repository. * Runs only on IBM computer systems. * Operating system is OS/2 based. - *Lacks third party support for PlantWorks/DAE.*
DEC	Consillieum/ BaseStar	* VAX based hardware. * Operating system is VMS based only. - *Limited vertical integration between BaseStar, which enables applications at the shop floor level, and the planning and management activites.* - *Lacks third party support.*
Hewlett Packard	Industrial Precision Tools (IPTs)	- *Geared only for shop floor software development.* - *No current integration between IPTs and third parties.* - *No vertical integration available yet between shop floor applications and the planning and management activities.*

Table 2-1 : Major commercial solutions with provision of an integrating infrastructure [AMR 1991]

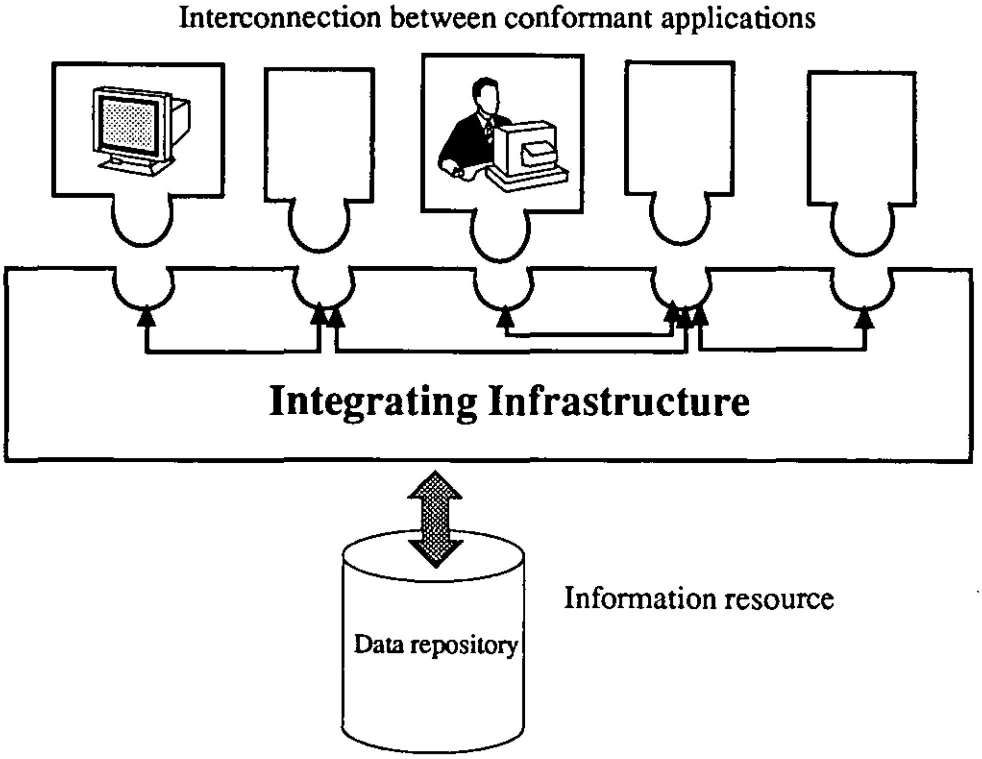

Interconnection between conformant applications
Integrating Infrastructure
Data repository
Information resource

Figure 2-4 : Interconnection via an integrating infrastructure

themselves need only have knowledge of how to use the IIS (i.e. NOT OF EACH OTHER).

An IIS is usually supported by software tools to help alleviate the complexities inherent in most systems integration projects [Timon *et al.* 1990, Hughes 1988]. A range of development tools for programmers and third party developers can be offered and may provide

- **Structured access to common integration services for**
  - inter-process communication;
  - information sharing and management via a data repository.
- **Consistent user and device interfaces to allow interaction over the IIS.**

Potentially, the use of an IIS

(i) *makes programming easier by insulating application software from complexities associated with managing of system resources.* This improves portability of application programs. For example, an application can run on different network types and can be referred to in a manner which is independent of physical location (i.e. it does not matter where that application resides).

(ii) *provides a data communication system that allows the building of an integrated system, comprising distributed software applications.* Once this system is built, the IIS enables the distributed applications to access the hardware resources in the system (which includes data repositories).

However, existing forms of IIS offer a restricted set of integration services and tools and are proprietary in nature. Here the IIS can enable a flexible mapping of software applications onto the physical resources of a system. This is of major advantage with respect to enabling change, the incremental extension of a system and providing a migration path from the use of legacy software and resources towards more interoperable components. They offer a view of the world and integration needs held by a particular system supplier. As a result, *they provide a limited 'de facto standard' interface capability where so called conformant software applications (which are strictly compatible to the IIS) will be supported.* Currently this limits the potential advantage gained from proprietary forms of IIS and is particularly limiting with respect to the inclusion of legacy systems [Singh and Weston 1993].

Furthermore, presently available forms of IIS do not treat application functions and information independently and separately. Thus *changes to either function or information will inevitably affect both because of their close dependency.* Indeed the task to effect the changes can be very demanding and disruptive to normal operations because careful consideration for enterprise-wide implications of such changes has to be given. This is due to the encompassing nature of the IIS towards promoting intra-organisation integration.

2.3 Information Reference Models

It is important to model or represent manufacturing enterprises [Lopes 1992, Zhang and Alting] in order to describe in a formal manner the ideal situation with regard to (i) information requirement and flow; and (ii) dependencies between activities. However, there is much diversity in the ideal situation in individual companies (even though they may be in the same industrial sector), in relation to the organisation structure adopted, operations carried out, manufacturing processes used, etc. Hence no two companies can be expected to be identical, each having idiosyncrasies and specific functional and information needs to realise its own business goals [St. Charles 1987]. As a result, it is recognised that manufacturing information is notoriously difficult to standardise. Notwithstanding these differences reference models are necessary to promote good practice and a certain degree of standardisation which can promote interoperation on an inter- or intra-company basis. *Where possible these reference models should capture and describe generic properties related to 'good practice' which are widely applicable. But they will need to be open to changes to allow modification and expansion to cater for customised needs* [Evans *et al.* 1993].

The reference model approach has been advocated and benefits demonstrated in previous research projects [Hars *et al.* 1992, Scheer 1991, Muhlemann *et al.* 1991], which include the ESPRIT project CODE (COmputer supported enterprise-wide Data Engineering). However, *to-date reference models reported in the literature have been specific and functional in nature, in as much that their use has been focused relatively sharply on a specific application domain,* such as integrated production planning development. For example, the use of a family of reference models was postulated by Scheer *et al.* [Scheer 1991, Hars *et al.* 1992, Hars 1990] which characterised specialised functions such as order entry, resource planning, management and scheduling. The models proposed were used successfully in facilitating requirement elicitation (this involving the identification and modelling of customisation requirements for applications) as well as enabling data re-engineering to suit specific user needs, as exemplified in Figure 2-5. Here the aim was to derive integrated production planning systems which are modular and reconfigurable in nature. *Generally the reference models were not conceived with an extended application domain in mind to characterise interoperation across conventional boundaries.*

Conversely, this thesis is focused on interoperation across functional boundaries where there is a need for reference models which comprise information of common interest to different functional areas and which can be shared by components of an MCS to enable them to functionally interoperate. Indeed the need for reference models which focus on interoperation across functional boundaries is becoming more widely recognised, where a noteworthy

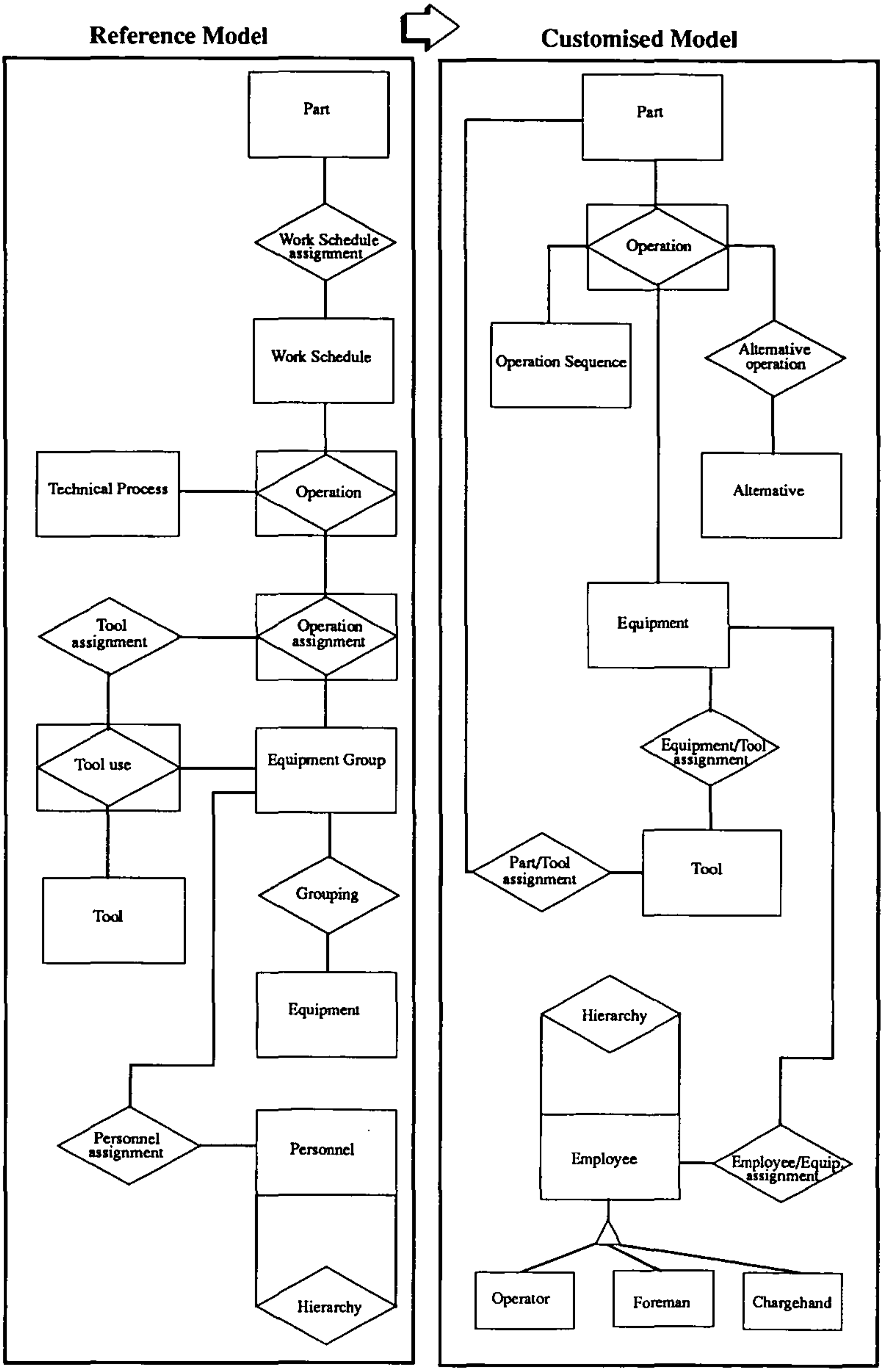


Figure 2-5 : Customising the production control area information need [Hars *et al.* 1992]

standardisation initiative known as MANDATE (MANufacturing DATa Exchange) has been recently setup to address the issues listed below (MANDATE [ISO 1991] is a working group of ISO's TC 184 namely ISO/TC184/SC4/WG8) :

- Model, form, and attributes of data exchanged between an industrial manufacturing company and its environment.
- Data to be used by manufacturing management for the purposes of managing the manufacturing company.
- Data controlling and monitoring the flow of materials within the company from a manufacturing management viewpoint.

Essentially information models of the MANDATE ilk promise to offer a degree of standardisation which can enable components of an MCS to functionally interoperate. However, standardisation efforts in this arena are in their infancy and will involve much discussion, investigation and deliberation before any consensus on standards emerge. Hence there is an immediate need for potential solutions to be capable of satisfying and to meet a representative set of needs which can feed into this standardisation work. Thereby it can further advanced as wider or complementary standardisation efforts mature.

2.4 System Design and Development

In reality integrated manufacturing systems must adapt and respond to changing needs. Further integration with other functions and re-engineering of existing functions will be inevitable for future enhancement or upgrading, as required, to refocus a business. Where possible, enhancement and modification of manufacturing systems should be enabled during each of the following life cycle phases with which system designers and builders, managers, engineers, operators and maintenance personnel are involved [Aguiar and Weston 1993b] :

- **Conceptual Design**

  In this life cycle phase the prime focus is deciding what a system should do. This can be achieved by analysing "as-is" (present) and "to-be" (potential) situations in order to identify means of achieving a set of improvement goals.

- **Detailed Design and Implementation**

  This involves specifying how the global requirements defined during conceptual design can be realised in terms of building the required solutions. Typically step-by-step implementation is achieved, with debugging of sub-systems carried out at each step.

- **Operation and Maintenance**

  This characterises the working life of the installed solution, as well as necessary adjustments and repair during the operational lifetime of the system. Generally speaking any major change will involve other upstream life cycle phases.

Commonly each life cycle phase is distinct in the sense that :

(i) normally different types of personnel with various perspectives and goals are responsible for each phase.

(ii) various methods and tools can be employed at each stage but seldom will their use be connected through common paradigms and system models [Motro 1987, Batini *et al.* 1986].

(iii) in view of (i) and (ii) "over the fence" system engineering is a common phenomenon with major discontinuities and misunderstandings as specifications and requirements for change traverse life cycle boundaries.

Thus current approaches to system design much reduces the opportunity to share and channel usable results and data produced in other phases. Consequently, realising life cycle support for an integrated system progressively through its design, implementation and run-time phases is by no means trivial, especially as the complexity of a given system grows. Many of the difficulty facing system designers and builders can be attributed to the *absence of a structured approach to creating systems which uses common formalisms to straddle the various life cycle phases* [Czernik and Quint 1992].

2.4.1 Structured Design and Modelling Methods to support Life Cycle Phases of an Integrated Manufacturing System

There are various structured design methods available which can support different life cycle phases of integrated manufacturing systems. Typically, they structure and represent certain aspects of the system under consideration; i.e. they provide a view or views of a system, related for example to function, information, behaviour, etc. [Orr *et al.* 1989]. Common used structured design methods include :

- **Entity-Relationship (E-R) Modelling.** This methodology was conceived to enable information modelling [Orr *et al.* 1989] and can systematically convert user requirements into a set of entity-relationship models [Jain *et al.* 1992]. Subsequently the E-R models defined can be used as the underlying model for a database management system. This can help facilitate information sharing in a more structured manner where the information model may encompass information which resides in a variety of data sources.

- **Yourdon** is a widely used process oriented methodology for designing software systems. It prescribes methods based on a set of diagrams (context, data flow, entity-relationship and state-transition) each of which illustrates a single perspective of the system [Weymont and Honeyager 1987]. Yourdon's structured design method has been used in a variety of applications. Also Yourdon and extensions to it have been combined with other software tools and used to design integrated systems [Savolainen 1991].
- **SSADM** (Structured System Analysis and Design Methodology) is a methodology originally conceived for software design [Cutts 1991]. It is widely used in commercial applications [Maji 1988]. The complete methodology encompasses six stages of a software project, viz: analysis, specification of requirements, selection of system options, logical data design, logical process design and physical design. To improve the input/output facilitates available to a system designer, it uses graphical modelling in the form of data flow diagrams and entity models.
- **IDEF** (U.S. Air Force ICAM - Integrated Computer Aided Manufacturing Definition) is a methodology derived from SADT (Structured Analysis and Design Technique) which has been more specifically tailored for use in manufacturing domains [ICAM 1985]. Currently IDEF comprises a suite of methods which essentially can be considered under one of the following main sub-divisions [Meta Software 1990] :

IDEF$_0$ - This is used to produce functional (or activity based) models of manufacturing systems or their sub-systems.

IDEF$_{1X}$ - This is a data modelling methodology used to describe entities and relationships between entities.

IDEF$_2$ - This is a dynamic modelling methodology that describes the time-variant behaviour of function blocks and information entities of a manufacturing system.

IDEF has been very widely used in a large number of industrial cases. It is used as a conceptual design modelling approach in many consultancy businesses around the world [Colquhoun *et al.* 1993]. A set of methods is currently under development [Mayer and Painter 1991], which includes Process Description Capture, Design Rationale Capture, Implementation Architecture Modelling, Organisation Modelling, and Three Schema Mapping Design. These new methods will further extend the scope of IDEF and hence its coverage of the life cycle of manufacturing systems.

- **GRAI** (Graphe à Résultats et Activités Interliés) is a methodology which was conceived to analyse and design production management systems [Akif and Documeings 1991}. It models function, structure and behaviour with the purpose of describing the flow of information, material and decisions in systems. It includes modelling views which represent time scales in the form of planning horizons and periods. On applying the methodology to a system, a graphical model is produced which relates activities, their time frame of operation, the decisions made and the information and resources required and used.
- **OOADM** (Object Oriented Analysis and Design Methodologies) is a collection of relatively new systems design methods which are based on the object oriented paradigm [Halladay and Wiebel 1993, Rumbaugh *et al.* 1991]. Already in many applications they have promised to replace conventional process oriented methodologies. Many object oriented design methods are reported in the literature that address one or more aspects of system design (either alone or combined with other methods) [Sanders *et al.* 1991, Hind *et al.* 1990, Schiel and Mistrik 1990, Jochem 1989, Terry and Matz 1989]. The main advantage of OOADM over process oriented approaches is the closeness of the object representation to the physical system being modelled [Bailin 1989], along with its orientation towards enabling simulation.
- **CIM-OSA** (Open Systems Architecture for CIM) has been proposed by AMICE (European CIM Architecture) consortium within the ESPRIT I and ESPRIT II programmes [Kosanke 1991, Jorysz and Vernadat 1990]. CIM-OSA comprises a methodology and a framework which embraces the specification of an integrating infrastructure [Aguiar and Weston 1993b]. It is suggested by certain authors that CIM-OSA 'goes far beyond previous modelling methodologies' and aims to support the design of CIM systems from their requirements definition (early stages of Conceptual Design) to their operation and maintenance [CIM-OSA 1989]. With CIM-OSA it is also claimed that a processable model of the CIM system can be produced as opposed to SADT-based methods which only produce static models and lack a dynamic modelling capability.

Various studies are reported in the literature which compare the capabilities of different modelling methodologies. To date, no one methodology includes capabilities for modelling the functional, information, dynamic and decision-making aspects of systems [Wyatt and Al-Maliki 1990, Wood and Johnson 1989]. As a result, *independent and separate use of a number of methods will be required if the formal modelling of systems is required on a comprehensive basis.*

2.4.2 Entry Point for Integrated Life Cycle Support

In summary, the application of most currently available design and modelling methodologies is primarily confined to the conceptual design phase, with a few extended to include limited support for the implementation phase as well. Functional, information and behaviour analysis is carried out with the aim of meeting a set of previously defined requirements and goals for the system concerned. Typically the static functional and information models generated using these methods will include formal definition and representation of (i) dependency relationships, and (ii) configuration and composition (e.g. database schema representation and entity-attributes, resource requirements to achieve the required functions). Having obtained models of the system the effect of changes and variations can be scrutinized. Thus possible system enhancements can be identified and represented by the models.

Hence the formal modelling of systems can provide an entry point for supporting the life cycle of manufacturing systems where *the models created (of function and information aspects) can serve as a source of knowledge during different life cycle phases*. However, in realising this potential it is necessary to :

- **Develop additional life cycle support tools coupled closely to the modelling tool.** Such a software toolset should exploit the knowledge contained within the model in order to ensure compatibility and continuity between life cycle phases, i.e. maintain consistency between the models produced and used during each life cycle phase [Singh and Weston 1994b].

2.5 Means of Controlling System Behaviour

In an integrated system, functions are coupled together through sharing of common information. Formal definition of interaction between the functional components of an integrated system requires clear and accurate descriptions of (i) the flow of information between function blocks and (ii) the form and type of information, which should be made available to support and drive those functions so that they can realise their assigned tasks [Singh and Weston 1994a]. Any lack of clarity very often gives rise to serious problems during system design, development, operation and changes. Hence when designing and implementing an integrated system, association between functions and information as well as functional dependencies must be well defined and clearly captured. If this can be achieved, the relationships defined can be used to determine and govern the manner in which the system behaves, particularly during run-time.

2.5.1 Association between Functional and Information Entities

As indicated in the literature there is a need to formally maintain an association between the functions carried out in a manufacturing system and the information entities they generate, access and manipulate [Scheer 1991, Shunk *et al.* 1986]. However, in manufacturing systems such associations are only formally maintained during the requirements definition and design stages of systems specification which in the software design process correspond to early phases of the development life cycle. To address this deficiency the following research proposals have been advanced :

- The "Y-CIM" model proposed by Scheer [1991], as illustrated in Figure 2-6, provides an integral organisational view of the different subystems of the enterprise. From such a view, the necessary links to be established between the different isolated subsystems making the exchange of information possible can be derived. Hence the model attempts to capture and represent the information needs of associated functions.

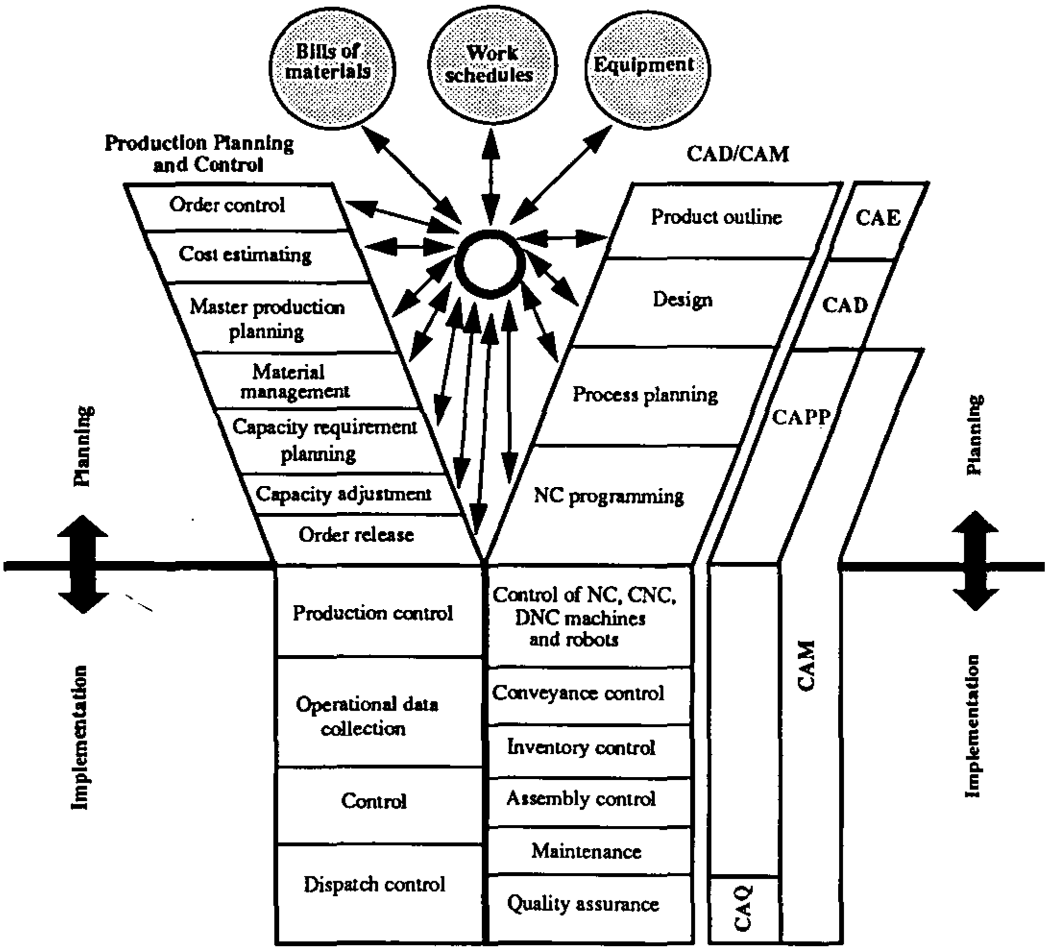


Figure 2-6 : "Y-CIM" model to promote integration of functions and information [Scheer 1991]

- The Triple Diagonal concept [Shunk *et al.* 1986], which is based on the use of IDEF$_0$, proposes a modification to the functional modelling approach and this includes a definition of information, control and material flows. Components of a manufacturing enterprise are classified and related to each other via a defined layered architectural relationship. Thus by including a definition of information resource requirements as well as material, information and control flows into the IDEF$_0$ functional model, a formal association between functions and information can be defined, with the input and output of information from associated functions being clearly identified, as illustrated in the example model of Figure 2-7.

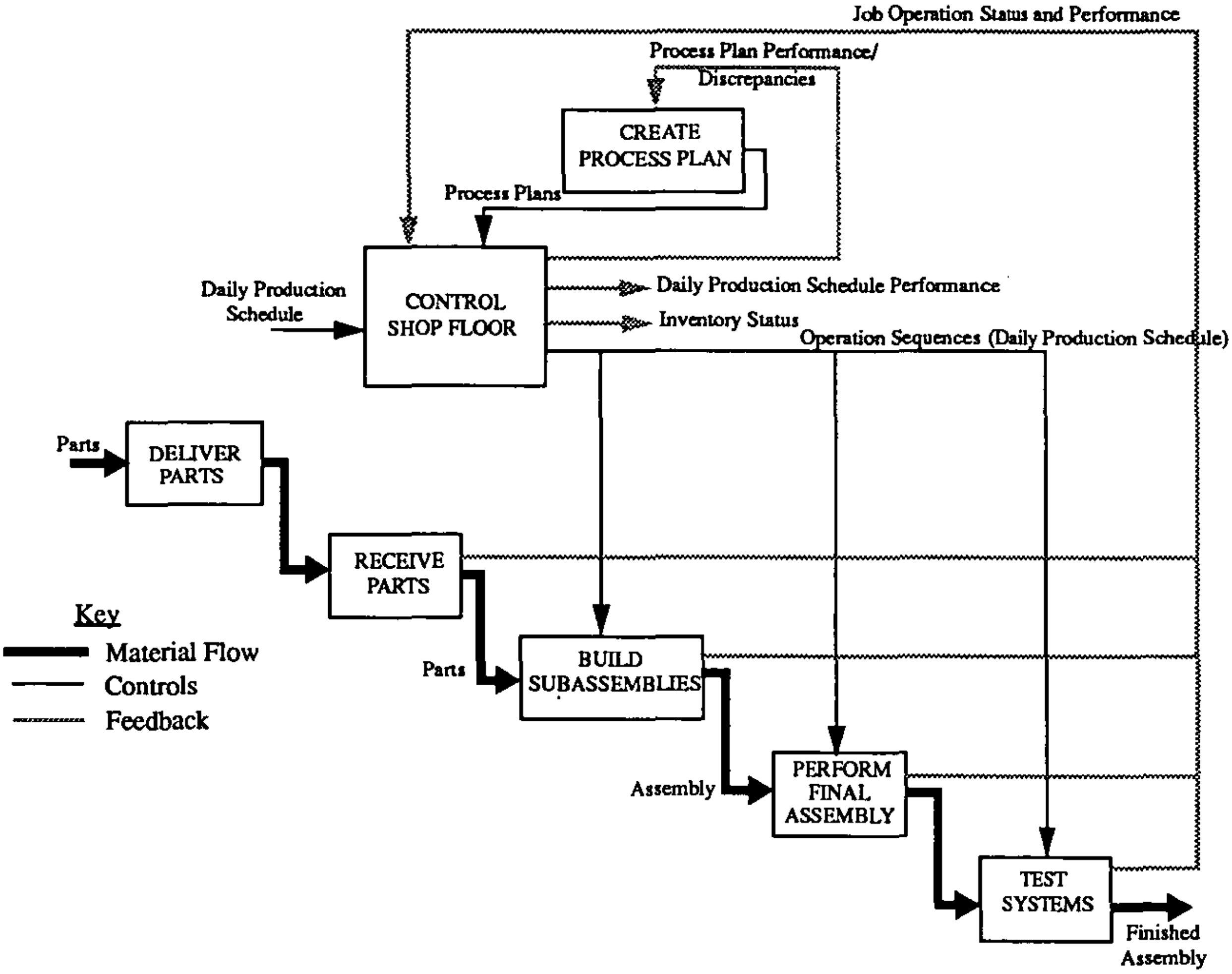


Figure 2-7 : Triple Diagonal (Material Flow/Controls/Information Integration) modelling [Shunk *et al.* 1986]

These formally defined associations can be made available for use in downstream life cycle phases of a system. However, use of an appropriate modelling method alone is insufficient to ensure that a particular function or information entity exists as part of an integrated system. This is because function and information entities are normally viewed separately, thus making it rather difficult to consider their close associations and dependencies. Thus there is a *need to establish and maintain an association between the function and information model streams*, in a way that can aid system design and development (see section 2.4).

2.5.2 Model Enactment to formally describe System Behaviour

It is recognised that formal specifications produced during system design, leading to a conceptual (functional and their associated information) requirements specification, can prove useful in determining and realising the required behaviour of a system. Thus it is necessary to [Singh and Weston 1994a]

(a) unify the perspectives of functional and information modelling; and

(b) facilitate the sharing and channelling results obtained during conceptual requirement definition so that they are useful for downstream life cycle processes, for example, to aid implementation and configuration with relation to the co-ordination and control of functional interaction between a given set of manufacturing components.

If such a capability can be formally realised using a modelling method coupled closely with a system design tool or set of tools, it will ensure consistency of results between life cycle stages (i.e. between system design and implementation stages). Such a capability can be referred to as model enactment where a conceptual functional requirements definition is used as a framework for more detailed behavioural modelling and implementation of that behaviour in a running system. This can be viewed as the process of enacting functional models. Such an enactment capability should inherit the following benefits :

(i) an appropriate formal definition of interactions between functional components based on
- functional dependencies derived from higher level system descriptions; and
- a description of data requirements to support the functions concerned.

(ii) defined means of supporting functional interaction management based on definitions and relationships established at a higher level, thereby facilitating control system behaviour (via suitable mechanisms) by providing a description of how run-time activities can be effectively coordinated.

However, in the following sections focus will only be on issues related to (a). Thus there is a need to enable both (a) and (b) via some methodology, which as later explained, could take the form of a software tool or tools.

Chapter 3

Achieving Interoperability

3.1 Research Focus

The author recognises that the sharing of common data, to enable a bonding between MCS components, constitutes one step towards fully achieving software interoperability [Singh and Weston 1993, Hars 1990]. In order ***that the benefits of software interoperability can be more fully realised, the next crucial step is to address problems of functional interaction (and the underlying issues of behavioural interaction) between MCS components. Thus the processes of co-ordinating and synchronising functional interdependencies and association between MCS components (with accountability for the shared data) need to be carefully managed and controlled.*** In practice, this is necessary to ensure and maintain discipline and harmony, to enable cooperation among interoperating software components [Hars 1990] and to establish well defined communication channels which can collectively promote and enhance intra-organisation interaction and co-ordination of activities [Scheer 1991].

In this thesis a novel approach to achieving software interoperability is conceived and advanced which offers means of (i) overcoming various inherent deficiencies and constraints which would severely inhibit or complicate MCS functional interaction, and (ii) tackling in a structured way integration problems associated with current forms of MCS and their component elements. A particular focus is on seeking an innovative methodology which can improve the reconfigurability of interoperating MCS software over the life cycle of such systems, thereby facilitating their adaptability in response to changing needs (for example, further integration with other functions and re-engineering of existing functions in order to modify and enhance the system's functionality). Through improving the adaptability of an MCS it should become possible to mitigate against early obsolescence and allow it to be more universally applied.

The eight years of industrial experience gained by the author, particularly in the precision machining industry has provided an important backcloth to this work. They have provided invaluable insight to (a) the practical problems faced by companies in trying to achieve applications integration, and (b) an understanding of "gaps" in technology and "know-how" which need to be filled to resolve such problems. This experience has been gained as follows :

- Between 1983 to 1986, as a practising manufacturing engineer the author was responsible for production work at ASEA Brown Boveri in Singapore. This is a medium sized company manufacturing tool and die components as well as plastic injection moulds. Shop floor experience was gained during this period of time as a CNC programmer and machinist, production and process planner, and production supervisor.
- Between 1989 to 1992, served as the Head of the Manufacturing Software Section of the Singapore Economic Development Board. The author was responsible for applied research and development activities, this to promote the general adoption of automation and computerisation in Singapore's manufacturing industry; particularly in relation to CIM, FMS, robotics and the application of expert and knowledge based systems [Singh 1992, Foong *et al.* 1992, Singh 1991]. This activity was centred on consultancy, manpower training and joint collaboration with local manufacturing companies and vendors.

Thus this research study has sought to adopt a mixture of pragmatic and formal approaches to resolving software interoperability issues. Focus is on enabling software interoperability in the arena of production planning which is defined as encompassing the management of flows of materials and goods as well as seeking to ensure capacity utilization based on customer orders and/or demand forecasts [Vollmann *et al.* 1988]; this includes order entry, resource planning and management as well as scheduling functions. However also considered are interoperability issues concerning MCS applications in other related manufacturing domains, which include the following :

- **Product Design**
- **Process Planning**
- **Finite Capacity Scheduling**
- **Cell Control**

MCS applications in the arena of finite capacity scheduling are responsible for the short term planning of manufacturing orders in a manner which optimises manufacturing operations on the shop floor. The need to improve interoperation via use of semi-automated and computerised cell control and production planning systems is considered essential in this study because of the current lack of uniformity normally found between proprietary systems used for production planning and those responsible for facilitating execution of plans; this is necessary to ensure that appropriate plans are effectively translated into actual production cycles. This lack of conformity has presented major problems for manufacturers, particularly in not encouraging a coherent view of both planned and actual information (derived from shop floor feedback) throughout the enterprise [Waterlow and Monniott 1986].

However, this research study is not aimed to place in the foreground details of necessary functional improvements to individual MCS functions. Rather it has sought to amplify the importance of applications integration and hence software interoperability. However, at times it does elaborate on the implications of these principles with respect to functional demands on individual components.

3.2 Objectives

Hence the overall objectives of the author's PhD study have been :

- to identify and specify architectural models of system functionality and information which themselves are based on studies of the inter-dependency of functions and commonality of information shared between production planning, product design, process planning, finite capacity scheduling and cell control processes.
- to address key issues of managing functional interaction, i.e. to study means of co-ordinating and synchronizing MCS functions. This by enabling and managing the interoperation of associated software applications in a flexible manner.
- to provide a formalised and structured methodology which can cope with high levels of complexity and change, straddling design, implementation, run-time and maintenance life cycle phases of interoperable systems. This to enable overall system reconfigurability, more optimal system design and operation and a reduction in the time and effort involved in creating such systems.

The emphasis of the study is on providing means of building "soft" rather than "hard" integrated solutions. Key to the methods derived will be a structuring of implementation processes based on the use of an integrating infrastructure which embodies common integration services. These common services will facilitate data management, access, manipulation and presentation, and support inter-process communication between conforming applications.

3.3 Production Planning as the Nucleus

In this study, the choice of *production planning information* as the initial nucleus of a model manufacturing information is a pragmatic one. It is *viewed as providing a comprehensive information system* that offers a large pool of manufacturing and logistical data *which bears varying degrees of commonality, interdependency and close association* [Hodgson and Waterlow 1992, Singh 1991, Harhalakis *et al.* 1990, Schnur 1987, Saxe 1985] *to that of other applications.* For example, bill of materials (BOM) and process planning information are

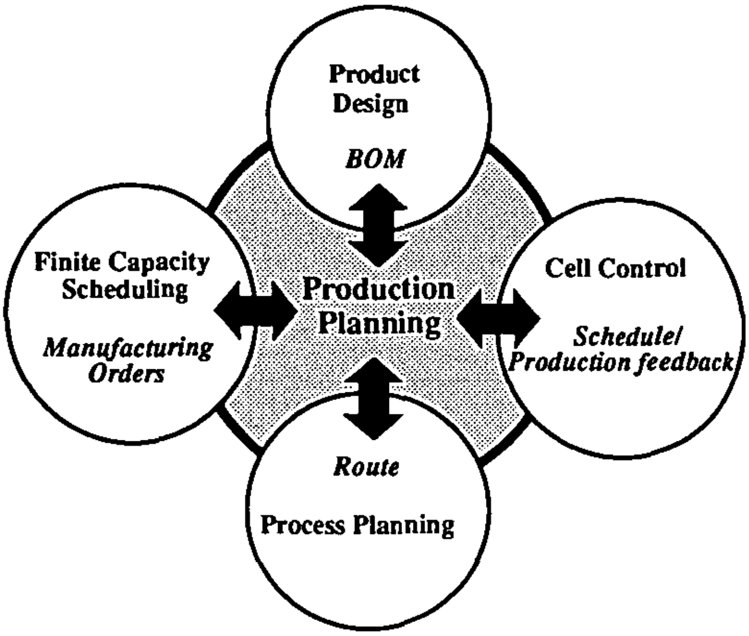


Figure 3-1 : Production planning information as the initial nucleus

common information entities viewed from different perspectives by CAD/CAM [CIM Strategies 1991, Lang-Lendroff and Unterburg 1989, Bohse and Harhalakis 1987] and CAPP [Ssemakula 1987, Logan 1986] (see Figure 3-1). This provides a basic means of establishing data repositories to facilitate information sharing with other functional areas within a given company.

3.3.1 Manufacturing Methods and Information Requirements

It is interesting to note that *the information requirements of adopted manufacturing methods, which include MRP II, OPT and JIT, demonstrate many similarities* [Singh and Weston 1993, Bond 1993, Plenert 1993, Lee 1993], *differing mainly in their emphasis and degree of focus on the activities concerned.* This is illustrated in the following :

- **OPT** is a computerised scheduling system which employs a standard MRP style database of **BOM** (bill of materials), **resources, routes** (including data on setup and operation times), **inventory** (raw material, WIP and finished product) and **demand** (specified by due date and quantity required).
- **JIT**, with its cellular manufacturing approach and the use of a 'demand pull' concept to control the production and movement of parts through the production process, requires information on **production schedules** (specified by start and finish dates on a daily basis), **BOM, inventory** (specified by lead-time for raw material and components delivery), **routes** (includes data on cell output, setup, process and cycle times) and **capacity** (specified by available working capacity for loading parts for manufacture).

The manufacturing method adopted in a given company will significantly influence the functional requirements of an MCS. Similarly characteristic properties of each manufacturing environment will directly influence these needs. This gives rise to considerable diversity among the functional properties of different production planning methods and systems which currently exist to support the manufacturing continuum (as highlighted in section 1.2.2). Hence when looking for similarities, it is more appropriate to consider and focus on similarities between different forms of MCS with regard to their information requirement (as indicated previously).

Indeed in this study an information model which represents information entities and their inter-relationships of common interest to different MCS functions, is identified and specified. This has been conceived to constitute essential production planning information of prime concern.

3.4 The need for MCS Interconnection and Interoperation

MCS software components are required to be interconnected in an effective manner before their interoperation can be enabled. Thus interconnection and interoperation between MCS software components are closely linked. A low level data inter-communication and information transfer facility (which facilitates data transfer between MCS components over a digital link) is an important prerequisite to efficiently interconnect MCS software components. However, generally such a digital data transfer capability needs to be built upon in order to facilitate interoperation in a controlled and deterministic manner.

This enhancement can be realised through the use of an integrating infrastructure (IIS) whose purpose is to provide structured access to information services in a way which simplifies interconnection between the component elements of software systems (see section 2.2.3 for further details). Ideally an IIS should comprise the following two levels of integration mechanisms and tools [Weston 1993] :

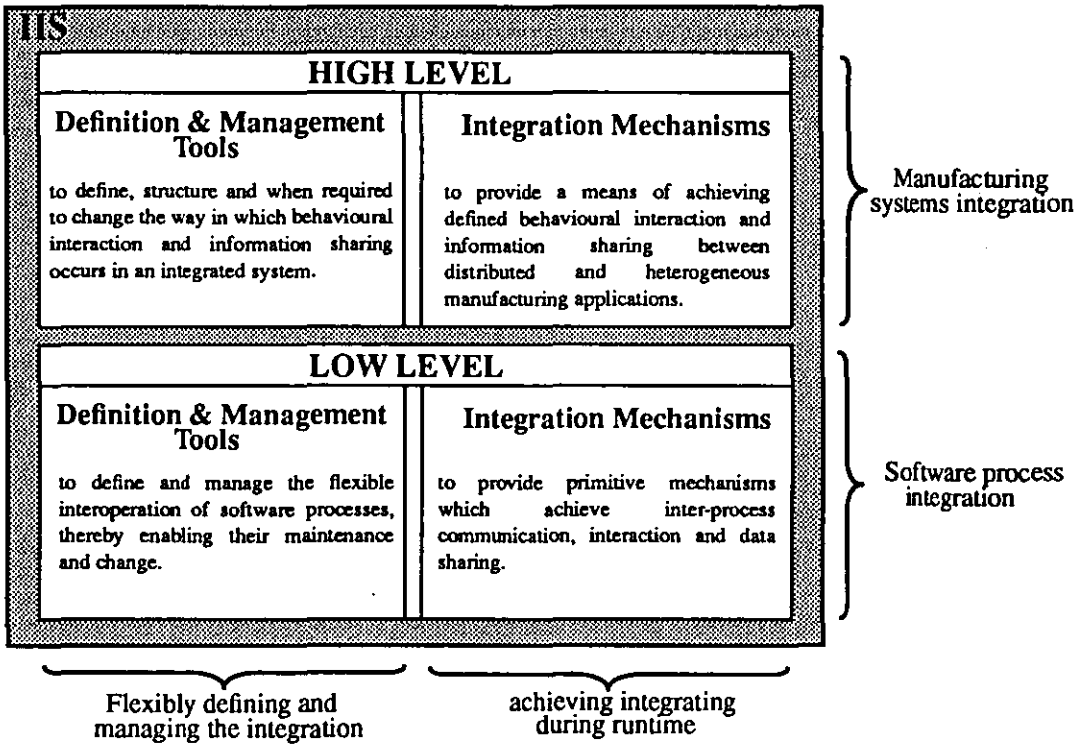
- **Low level**

  This can encompass a number of general purpose means of accomplishing the integrated operation of computer software processes (or software applications). In manufacturing enterprises (as in many other computational systems) software applications will be embedded in equipment and computer systems. Hence the low level mechanisms need to resolve differences arising from heterogeneity in computer processing hardware, software, operating systems, networks, human interface systems and data sources supported. They will be required to support appropriate low level protocol between interacting software applications as well as to resolve differences in representing and storing information.

- **High level**

  This includes high level integration mechanisms and system management tools to more directly facilitate the interoperation of MCS software components (which is one of the primary issues being addressed in this research study). They embody domain knowledge (relating more specifically to manufacturing systems integration) to enable the integrated operation of manufacturing applications and their internal threads of application functionality. However, the high level mechanisms need to be built upon their low-level counterparts [Weston 1993].

Figure 3-2 generalises the distinction between low level IIS mechanisms and tools and their high level counterparts.


IIS
HIGH LEVEL
Manufacturing systems integration
LOW LEVEL
Software process integration
Flexibly defining and managing the integration
achieving integrating during runtime

Definition & Management Tools

to define, structure and when required to change the way in which behavioural interaction and information sharing occurs in an integrated system.

Integration Mechanisms

to provide a means of achieving defined behavioural interaction and information sharing between distributed and heterogeneous manufacturing applications.

Definition & Management Tools

to define and manage the flexible interoperation of software processes, thereby enabling their maintenance and change.

Integration Mechanisms

to provide primitive mechanisms which achieve inter-process communication, interaction and data sharing.

Figure 3-2 : Components of an Integrating Infrastructure [Weston 1993]

A key aspect of this research study has been a structuring of MCS implementation processes based on the use of an IIS.

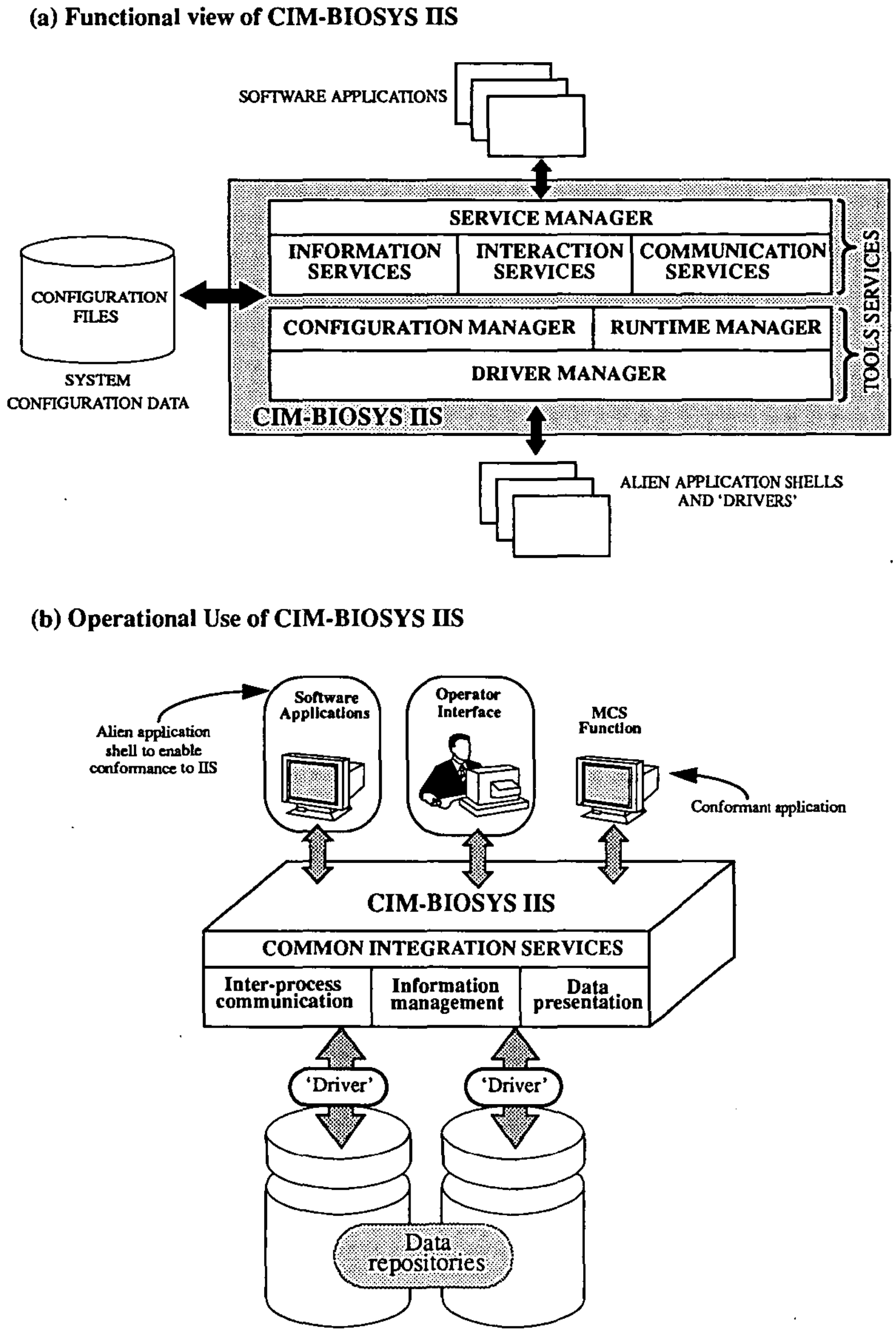
3.4.1 The CIM-BIOSYS Integrating Infrastructure

Since 1986, the general requirements of integrating infrastructures for manufacturing systems integration have been studied by researchers of the MSI[1] Research Institute at Loughborough University of Technology. The author's research has also contributed to this study. In 1990, MSI research led to the development of the CIM-BIOSYS (CIM-Building Integrated Open SYStems) IIS (Integrating Infrastructure) which, as depicted in Figure 3-3, achieves a unification of general purpose computational integration mechanisms and tools and has been used to create a variety of 'proof-of-concept' and 'live industrial' integrated systems [Weston 1993]. It is configured to operate in a distributed manner under the UNIX environment over a network of SUN computer workstations. This IIS provides a means for structuring, decomposing and simplifying solutions and supporting their run-time execution and change. Of particular importance has been an implicit ability to build and modify systems (including systems of very wide scope) on an incremental basis. The use of the CIM-BIOSYS IIS has demonstrated significant savings in the cost and time involved in manufacturing integration projects [SI Group 1994].

CIM-BIOSYS II offers important advantages over contemporary turnkey and custom built integration methods, in that inherently it :

- **Deals with complexity**
  Applications only need knowledge of how to access the platform, rather than how to access 'n-1' other applications within the integrated system. This results in a vitally important means of coping with increased complexity as the system complexity will grow in proportion to the number of applications rather than the square law fashion found using contemporary approaches.
- **Copes with change**
  It removes integration knowledge from interacting applications concerning the actual structural relationships, interaction mechanisms, information structures, data formats and communication protocol; the integrating infrastructure deals with such issues. This knowledge is placed in the form of configuration data which can be used in a systematic way to enable and support change.
- **Promotes standardisation**
  This is achieved by specifying a consistent interface between the services of the integrating infrastructure and the applications which use them. Also the integrating infrastructure is itself decomposed into more manageable sub-systems which can be

[1] MSI - Manufacturing Systems Integration Research Institute based at Loughborough University.



(a) Functional view of CIM-BIOSYS IIS

(b) Operational Use of CIM-BIOSYS IIS

Figure 3-3 : CIM-BIOSYS Integrating Infrastructure

standardised or built on existing standard mechanisms and services. Thus applications can be treated essentially as open applications and as such they themselves can become standard building blocks of systems.

MSI researchers had also previously identified and produced methods and software tools for dealing with certain classes of non-conformant (or alien) applications. Here the term 'alien' is used to imply that the application component (which may be a software package or software embedded in a machine control system) is not inherently compatible with the CIM-BIOSYS IIS architecture. 'Drivers' and 'alien application shells' represent the particular software tools referred to here which respectively provide MCS resources (such as databases and datafile systems) and legacy application components with sufficient capability that they can use the integration services of the CIM-BIOSYS IIS. These methods and tools are essential in order to allow for the inclusion of embedded legacy systems thus helping to safeguard the user's existing investment in computer systems [Hollyman and Anderson 1991]. A summary of some of the major differences and similarities between CIM-BIOSYS IIS and other commercially available solutions with provision of an IIS is provided in Table 3-1.

It must be stressed at this juncture that CIM-BIOSYS IIS includes only low level integration mechanisms and tools, as depicted in Figure 3-2. It embodies common integration services. These common services facilitate data management, access, manipulation and presentation, and support inter-process communication between MCS software components. Thus the CIM-BIOSYS IIS was chosen as a primary building block in this research study, thereby providing a foundation for implementing and evaluating high level integration mechanisms and tools that have been developed in this research study to facilitate software interoperability. In isolation, the CIM-BIOSYS IIS can only facilitate bottom-up system build; hence by building the high level mechanisms upon its low level counterparts, top-down system design and construction can be more readily facilitated.

			Comparison to CIM-BIOSYS IIS	
Vendor	**Product**	**Remarks**	**Similarity and common emphasis**	**Limitations of commercial solutions**
IBM	PlantWorks/DAE DAE - Distributed Automation Edition	*- Proprietary in terms of the following :* * Need to comply with IBM's SAA based strategy of a CIM repository. * Runs only on IBM computer systems. * Operating system is OS/2 based. *- Lacks third party support for PlantWorks/DAE.*	***1) Provision of tools to facilitate integration*** A range of development tools for programmers and third party developers to facilitate integration of MCS software applications are normally offered for the following : **•Consistent user and device interface.** **•Access to common integration services.** - Information sharing and management - inter-process communications ***2) Integrating infrastructure*** • to simplify, structure and service interconnection between MCS software applications. • to provide structured access to common integration services for communication and information management.	***1) Function & information coupling*** Functions and information are not decoupled and treated separately which creates the following difficulties : **•Information Access** Information is embedded and exists exclusively for the sole use of the specific application. **•Function & information changes** Any changes to either function or information will inevitably affect both because of their close dependency. ***2) Support for conformant applications*** Only conformant applications which are strictly compatible to the integrating infrastructure are supported. This poses serious problems for inclusion of "as is" (or legacy) systems.
DEC	Consillieum/ BaseStar	*- Proprietary in terms of the following :* * VAX based hardware. * Operating system is VMS based only. *- Limited vertical integration between BaseStar, which enables applications at the shop floor level, and the planning and management activites.* *- Lacks third party support.*		
Hewlett Packard	Industrial Precision Tools (IPTs)	*- Geared only for shop floor software development.* *- No current integration between IPTs and third parties.* *- No vertical integration available yet between shop floor applications and the planning and management activities.*		

Table 3-1 : Summary of major commercially available solutions with provision of an integrating infrastructure.

3.5 An Overview of the Methodology Derived

A meta-level overview of the methodology adopted and developed by the author during this research study to enable interoperation of MCS components [SI Group 1994] is depicted in Figure 3-4. This methodology comprises five inter-related and consistent sub-methods which collectively structure and support key aspects of MCS design, build, operation and change management. *One of the underlying concepts adopted in the methodology is (as far as possible) to decouple MCS functions from their information repositories* so as to enable the information to be treated independently from the functional capabilities realised by software applications. This not only enables easier access to information but also decouples changes to application processes from those associated information systems. The purpose of the sub-methods are outlined as follows :

(I) MCS Specification

A set of high level modelling methods, based on a set of generic reference models, is used to facilitate MCS design. The output consists of particular models of 'MCS functions' and 'MCS information entities and their interrelationships'.

(II) Means of Enacting Function Models

A set of build tools are used to create executable descriptions of the system behaviour where the descriptions are consistent with the MCS function models generated by (I). The output descriptions can be used to control the way in which MCS components interact during system run-time [Singh and Weston 1994a].

(III) Means of Enacting Information Models

A second set of build tools are used to create and populate information models in a form which structures and enables control of information shared between MCS components during system operation [Singh and Weston 1994b]. The output descriptions are consistent with the MCS information models created in (I) and with the outputs generated from (II).

(IV) Use of an Integrating Infrastructure (IIS)

This facilitates MCS run-time operation in a flexible data-driven manner, mapping distributed software solutions onto physical resources. Here the integrating infrastructure (IIS) is charged with resolving differences in the physical system relating to heterogeneity, distribution and data fragmentation [Weston 1993]. Indeed the use of the IIS is the key to the methodology. As the IIS assumes responsibility for maintaining knowledge of integration details (such as the networks used, the hardware and operating systems on which an MCS component is run, the location of an information fragment, etc.), the MCS components themselves need only have

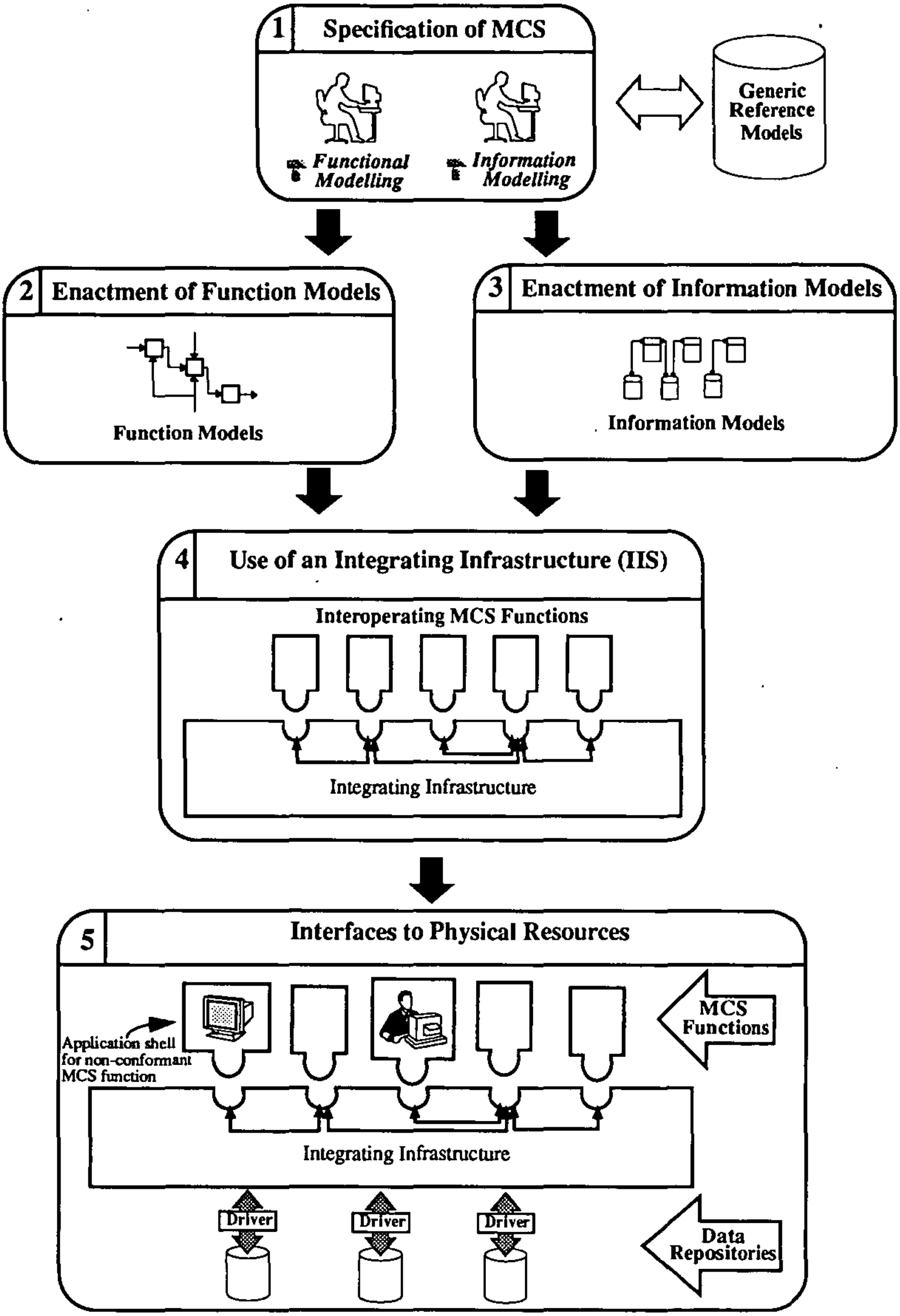

1 Specification of MCS
Functional Modelling
Information Modelling
Generic Reference Models
2 Enactment of Function Models
Function Models
3 Enactment of Information Models
Information Models
4 Use of an Integrating Infrastructure (IIS)
Interoperating MCS Functions
Integrating Infrastructure
5 Interfaces to Physical Resources
Application shell for non-conformant MCS function
MCS Functions
Integrating Infrastructure
Driver
Driver
Driver
Data Repositories

Figure 3-4 : Overview of Methodology

knowledge of how to use the IIS (i.e. NOT OF EACH OTHER). Essentially this leads to a linear relationship between system scope (in terms of the number N of MCS components) and complexity as opposed to the square law relationship inherent in pair-wise integration methods.

(V) Interfaces to Physical Resources

An essential element of this methodology is the ability to flexibly map software applications onto system resources, i.e. databases and computer hardware. 'Drivers' and 'alien application shells' represent the software tools created to bring MCS resources and components (which include proprietary software packages, database and datafile systems) to a level of conformance which enables interoperation over an IIS. This provides a migration path towards more 'open' components at a later stage.

Chapter 4

Information Architecture for MCS

4.1 General Considerations

It is important to model or represent manufacturing enterprises [Lopes 1992, Zhang and Alting] in order to :

- Well define (i) functions and activities in terms of their associated inputs and outputs, (ii) inter-dependencies and close relationships between functions, and (iii) information requirement and flow as seen from the view point of a systems designer.
- Accurately capture the reality of business goals and their relationships with manufacturing tasks.

The author's previous experience in the successful development of a working CIM model, which is still used to provide a technological showcase for the precision machining industry for discrete part manufacture [Singh 1992, Foong *et al.* 1992, Singh 1991], served as a useful and valuable source of reference, insight and input to this research work. The CIM model project involved close collaboration between vendors and industrial end users. It captured many of their key needs and led to an accepted and intrinsic representation of activities to support part manufacture. This can be universally useful in understanding important aspects of interoperability in a manufacturing enterprise. In particular it has served to

- Identify a functional model which is representative of precision machining enterprises.
- Globally specify information inputs and outputs for MCS functions.
- Ascertain which computer-aided tools are available and which must be tailored or developed to enable and enhance integration processes.

The reader can refer to Figure 4-1 for an overview of the functional properties and information entities of this CIM model and Appendix II for further details.

There have been a number of alternative models conceived to define the functionality of CIM systems where often the studies have taken different modelling perspectives [Paranuk 1988, Yeomans 1986]. The ESPRIT CIM-OSA (CIM-Open System Architecture) consortium [CIM-OSA 1989] defined a standard for CIM implementation, offering an 'enterprise wide framework' which can structure interactions between people and machines as well as define relationships with traditional data processing systems. The CIM-OSA "top-down" approach to enterprise integration can work well at the conceptual design level of integration projects, i.e.

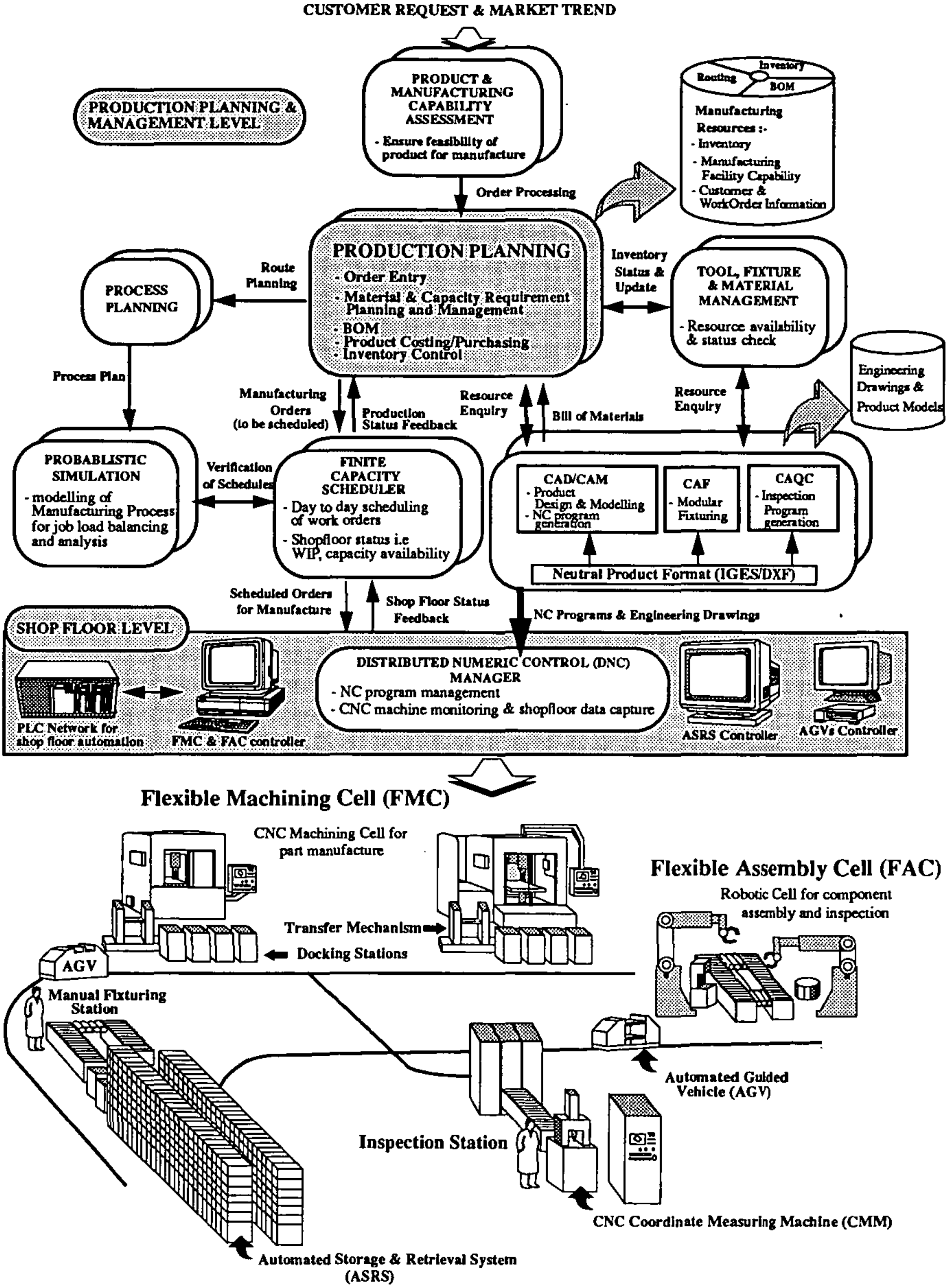

CUSTOMER REQUEST & MARKET TREND
PRODUCTION PLANNING & MANAGEMENT LEVEL
PRODUCT & MANUFACTURING CAPABILITY ASSESSMENT
- Ensure feasibility of product for manufacture
Order Processing
Routing
Inventory
BOM
Manufacturing Resources :-
- Inventory
- Manufacturing Facility Capability
- Customer & WorkOrder Information
PRODUCTION PLANNING
- Order Entry
- Material & Capacity Requirement Planning and Management
- BOM
- Product Costing/Purchasing
- Inventory Control
Route Planning
PROCESS PLANNING
Inventory Status & Update
TOOL, FIXTURE & MATERIAL MANAGEMENT
- Resource availability & status check
Engineering Drawings & Product Models
Process Plan
Manufacturing Orders (to be scheduled)
Production Status Feedback
Resource Enquiry
Bill of Materials
Resource Enquiry
PROBABLISTIC SIMULATION
- modelling of Manufacturing Process for job load balancing and analysis
Verification of Schedules
FINITE CAPACITY SCHEDULER
- Day to day scheduling of work orders
- Shopfloor status i.e WIP, capacity availability
CAD/CAM
- Product Design & Modelling
- NC program generation
CAF
- Modular Fixturing
CAQC
- Inspection Program generation
Neutral Product Format (IGES/DXF)
Scheduled Orders for Manufacture
Shop Floor Status Feedback
NC Programs & Engineering Drawings
SHOP FLOOR LEVEL
DISTRIBUTED NUMERIC CONTROL (DNC) MANAGER
- NC program management
- CNC machine monitoring & shopfloor data capture
PLC Network for shop floor automation
FMC & FAC controller
ASRS Controller
AGVs Controller
Flexible Machining Cell (FMC)
CNC Machining Cell for part manufacture
Transfer Mechanism
Docking Stations
AGV
Manual Fixturing Station
Flexible Assembly Cell (FAC)
Robotic Cell for component assembly and inspection
Automated Guided Vehicle (AGV)
Inspection Station
CNC Coordinate Measuring Machine (CMM)
Automated Storage & Retrieval System (ASRS)

Figure 4-1 : Overview of the functional and information network within the CIM Model

where 'what needs doing' is determined. The strategy maxims CIM-OSA offer can ultimately lead to support of the complete life-cycle of manufacturing enterprises, including their component MCS [Aguiar and Weston 1993a, Schonewolf *et al.* 1992]. This research adopts a more pragmatic approach to system design but it builds upon the CIM-OSA concept of *partial models which embody knowledge concerning 'good practice' or 'good solutions' within an enterprise.* These partial models can be reused in building and updating various new systems, thereby reducing the time involved and improving the quality of the resulting manufacturing system.

Within the scope of this research study, information models have been defined which encompass essential information of common interest and can be used as a generalised resource in many discrete parts manufacturing environments (discussed later, in section 4.2). Hopkinson, in his analysis of user needs relating to information standards, strongly stressed the point that :

> " *What matters most of all for the user is the information the system holds; the way in which it is held and accessed, and what can be done with it, are also important but **the means are of no value if the information itself is not what is needed*** " [Evans *et al.* 1993]

The models defined in this study represent shared information which are "items of knowledge". These have global interest to the MCS system concerned, i.e. they can be considered to be CIM-OSA partial models. The use of partial models is essential as earlier studies [Lars 1990] have shown that in order to manufacture in a more rational, timely and cost effective manner, it is vital to capture and disseminate knowledge about the manufacturing enterprise. For example, a fundamental requirement during many Product Design processes is to ensure the manufacturability, ease of assembly and testability of the product. Therefore, it is necessary for the designer to have access to accurate information about the manufacturing process, with due consideration to constraints such as the availability and type of resources (e.g. material, toolings and fixtures) and orders which exist for the product. For example, information stored in conformance with the partial models will be useful in matching product specifications and requirements to the capabilities available in the enterprise. However, this fundamental requirement is not usually provided for, this being a major cause of a fairly large próportion of engineering changes and discrepancies in manufacturing enterprises today [Singh 1991, Lars 1990].

4.2 MCS Specification

For manufacturing information to be shared between MCS applications in an effective way, ideally it is necessary to make available a pool of most of the information entities that are commonly shared between two or more applications. In order to achieve this an information model is required which identifies real world objects, their key attributes and inter-relationships.

A detailed analysis of production planning functions, with regard to their commonality of information and functional inter-dependency with Product Design, Process Planning, Finite Capacity Scheduling and Cell Control processes, was carried out by the author. This involved the following areas of study and analysis :

(i) A review of the literature and current practice with regard to the techniques used to accomplish integration of production planning, CAD/CAM, CAPP and cell control systems. Some of the findings of this work are as summarised in Table 4-1.

Relevant references	Integrated systems	Commonality of Information	Functional Purpose & Dependency
Halevi and Weil 1992 CIM Strategies 1991 Scheer 1991 Singh 1991 Harhalakis et al. 1990 Lang-Lendroff *et al.* 1989 Scheer 1989 Schnur 1987 Bohse and Harhalakis 1987 Ssemulaka 1987 Logan 1986 Saxe 1985	**Integration of Production Planning with** **- CAD/CAM** **- CAPP** **- Shop floor control systems**	**Bill of Materials (BOM)** **Inventory & resources** **Process plans/Routes** **Schedules** **Shop floor status feedback**	1. Product hierarchical decomposition into sub-components to aid i) resource planning; ii) sub-components manufacture; and iii) standard components procurement. 2. Material requirement planning and allocation for part manufacture (e.g. raw material, toolings, fictures, manufacturing facilities, etc.) 3. Sequencing of manufacturing actiivites for part manufacture.
Zäpfel and Missbauer 1993 Lee 1993 Muhlemann *et al.* 1991 Scheer 1991 Ptak 1991 Waterlow and Monniott 1986 Luscombe 1991	**Integrated Production Planning with modularised functions**		Material requirement planning Resource allocation and scheduling Order entry Routing Capacity planning Inventory management Shop floor status monitoring & acquisition

Table 4-1 : Summary of literature review

(ii) By building on insights gained from the CIM model project [Singh 1992, Foong *et al.* 1992, Singh 1991]. In particular it has served to

- Globally specify common information inputs and outputs for MCS functions.
- Identify close relationships and dependencies among typical MCS components.
- Define data mapping requirements based on functional relationships.

A general overview of information shared between MCS functions and their information dependencies is illustrated in Figure 4-2. As a result of close collaboration with end users, the CIM model project was successful in capturing and reflecting their practical requirements, this in terms of shared information requirements between typical MCS components to facilitate their interoperation. This consideration (of the end users viewpoint) is very important in helping to validate the findings derived from (i), this being important in order to achieve a pragmatic and industrially acceptable solution.

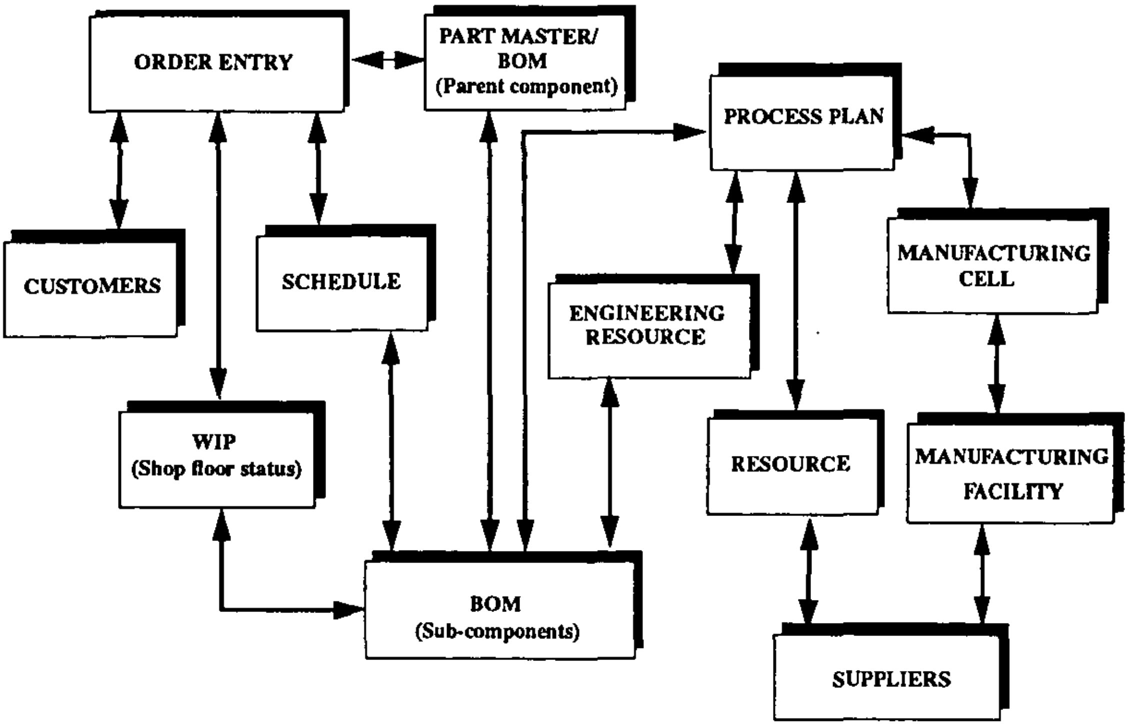


Figure 4-2 : Overview of information flow and data dependency

(iii) Through the examination of generic features and common attributes among representative commercially available computer-aided production management (CAPM) packages [Buyer's Guide Supplement 1990]. Those selected are listed in Table 4-2. The choice of CAPM products was made on the basis that (a) collectively they encapsulate the generic working knowledge of a number of vendors which itself reflects the perceived needs of many manufacturing user organisations; and (b) the technical support offered by the vendors concerned in understanding the information and functional properties of their products.

		CAPM Products									
		MANMAN	EZ_MRP	ELMS	CIMM	COMET	MFG/PRO	Fourth Shift	MCC		
CAPM Subsystems	Planning	✓	✓	✓	✓	✓	✓	✓	✓	Capacity Requirement Planning	Available Modules
		✓			✓	✓	✓	✓		Master Production Schedule	
		✓	✓	✓	✓	✓	✓	✓	✓	BOM	
		✓	✓		✓	✓	✓	✓	✓	Manufacturing order management	
		✓	✓	✓	✓	✓	✓	✓	✓	Routing	
	Inventory Management	✓	✓	✓	✓	✓	✓	✓	✓	Material Requirement Planning	
		✓	✓	✓	✓	✓	✓	✓	✓	Inventory Control	
	Scheduling	✓	✓		✓	✓	✓	✓	✓	Scheduling	
	Shop Floor	✓	✓		✓	✓	✓	✓		Shop floor control	
	Manufacturing Support services	✓	✓	✓	✓	✓	✓	✓	✓	Sales Order Processing	
		✓	✓	✓	✓	✓	✓	✓	✓	Purchasing	
		✓	✓		✓	✓	✓	✓		Financials	
		✓	✓	✓	✓	✓	✓	✓	✓	Management reporting	
		✓		✓	✓	✓	✓	✓		Costing	

Table 4-2 : CAPM packages examined

However, the author experienced inherent difficulties in formally evaluating and analysing these candidate systems on a common basis because of major differences in their underlying philosophies and their implicit understanding of the activities within a factory and how they should be controlled. This is further complicated by the proliferation of names used to refer to essentially similar and basic functional capabilities which collectively enable production management. Also in different systems different functions can be included under the same name. Thus the CAPM systems analysed were compared and classified with reference to the following subsystems [Maull and Childe 1993, Timon *et al.* 1990, De Toni *et al.* 1988]; this in order to gain a more structured and uniform understanding of the functions offered :

- Planning
- Inventory Management
- Scheduling
- Shop Floor
- Manufacturing Support services

Following this analysis common classes of MCS application module and information entity were identified, as listed in Table 4-3. As a result information models were identified and defined to satisfy generic requirements, these models being listed in Table 4-4.

	Production Planning
Product Design	* Inventory / Part master records * Bill of Materials (BOM)
Process Planning	* Process plans / Routes * Manufacturing facility records
Finite Capacity Scheduler	* Manufacturing orders * Bill of Materials (BOM) * Work centre capacities
Cell Control	* Scheduled manufacturing orders * Shop floor production and status feedback

Table 4-3 : Commonality of information

Manufacturing Facility	Information on manufacturing support facilities with data on manufacturing capabilities and specification.
Part Master/ BOM	Product structure according to its sub-components relationship.
Resource	Inventory record and status for raw materials, fixtures and tools inclusive of labour and facilities, i.e. work centres and processes.
Process Plan	Routing for part manufacture. It includes the sequence of operation for planned and alternative processes and manufacturing resource requirement.
Order Entry	Order registration for order type, quantity, batch size and due date.
Schedule	Production time-table where manufacturing orders are scheduled according to order commitment and availability of resources.
WIP	Shop floor status feedback, actual to planned comparison, and work centre utilisation rate.
Engineering Resource	It includes resources related to or are assigned for part manufacture, e.g. engineering drawings and NC programs.
Manufacturing Cell	Grouping of manufacturing stations for manufacture of a family of products.

Table 4-4 : Information Models

The reader should refer to Appendix III for further details on the information entities and attributes represented in the information models. Examples of associations between these information models and specific information models which form the basis of the MCC [MCC 1989] and ELMS [ELMS 1990] proprietary software packages are included in Appendix IV. A case study (discussed in detail in section 4.3) has been carried out in collaboration with the University of Bradford Management Centre to validate the applicability of the generic information models. In this case study, the ELMS CAPM software package has been re-engineered with reference to the generic reference models to facilitate its ease of further development to enhance its existing functionality.

As part of the author's research study, system design and modelling tools have been used to formally represent and structure the function and information models defined in tables 4-3 and 4-4. IDEF_0 was used to identify and define dependencies and inter-relationships between the common classes of MCS applications identified. In addition, IDEF_{1X} was used to represent entity-attribute relationships for information models and the data modelling language EXPRESS was used for information modelling. The choice and application of these software tools is further discussed in chapter 6.

It is important to stress at this juncture that the generic information models identified and formally defined as part of this research study, provide an important cornerstone of the author's overall approach to enabling software interoperability in the MCS domain. Furthermore, the author is confident that the components of those models and their interrelationships are appropriate, certainly with respect to their use for the forms of MCS investigated here. However, it is not argued that the models are sufficiently definitive or complete to form the basis of a standard model, but as illustrated in this thesis, they are sufficiently definitive and complete to contribute towards an important advance in creating more open and configurable forms of MCS and as such can provide a reference model of good practice which can be refined and enhanced, possibly until it reaches the status of a standard. Also later in this thesis it will become clearer that a second cornerstone of the author's approach is the use of model enactment to guide MCS life cycle processes (this being outlined in section 2.3). Indeed through the use of formal models and a set of tools which can manipulate and transform those models, improved opportunities exist to refine and enhance a reference model until it becomes more widely accepted and used.

4.2.1 Characteristics of the Information Model

The information models identified incorporate essential data (i.e. data of prime concern) that will very commonly be used in any discrete part manufacturing environment. Thus *they can serve as a foundation upon which the rest of the enterprise data can be built.* Together, these information models can provide the enterprise with a single coherent view of engineering, production and management information which will be in common usage throughout the product life cycle. MCS data will need to be stored with reference to these generic models and in a practical system will be stored in distributed data repositories to enable common access and usage by MCS components.

It is recognised that information is notoriously difficult to standardize [Evans *et al.* 1993]. Therefore, as outlined above the choice of information models is not meant to be fixed and exhaustive in nature, rather *they have been chosen to serve as generic reference models which are open to changes and can be modified and expanded when necessary.* Bearing these restrictions in mind the information models can be considered to possess the following characteristics:

- **Wide applicability**
  They conform to the requirements of many potential users and are not structured or geared towards a particular enterprise but rather for a set of enterprises. In this study the reference models where chosen to support the precision machining industry but they could well have an essential form which can constitute the basis of reference models for other industrial sectors.

- **Flexibility**
  They are adaptable and can be customised to specific needs of a user. The flexibility is attained as a result of their formalism in a computer readable form offering opportunities to manipulate their underlying data structures.

Also collectively these information models can serve as partial models which can be further expanded or coupled with specific information models that contain information which is unique to the manufacturing environment concerned. Certainly before the proof-of-concept models advanced here could form the basis of any standard it would be necessary to consider the extent to which a reference model should aim to be complete, as the inclusion of entities seldom used will inevitably lead to some overheads in terms of required data storage and processing capabilities.

4.2.2 Reference Models for an Extended Application Domain

The reference model approach adopted in this research study has an extended application domain in mind which crosses conventional product boundaries and indeed crosses common organisational boundaries found in many manufacturing enterprises. Thus the reference models specified in this research comprise information shared by several functional areas and effectively serve as a precursor to enable components of an MCS to functionally interoperate. In comparison, the approaches advocated and validated in previous research projects [Hars *et al.* 1992, Scheer 1991, Muhlemann *et al.* 1991] have been specific in nature, in as much that they have focused relatively sharply on an application domain, such as integrated production planning development (as highlighted in section 2.3).

The perspective and inputs gained from on-going standardisation initiatives, such as MANDATE [ISO 1991] to model manufacturing information (see section 2.3 for details), will undoubtedly help to further advance the generic reference models identified in this study. These reference models aim to provide an effective solution capable of satisfying current needs.

4.3 Application of Generic Reference Models - A Case Study

In 1990 the University of Bradford Management Centre (UBMC), under the sponsorship of ACME, was responsible for the development of generic CAPM software which seeks to address the key production management needs of SMEs [Afferson *et al.* 1992, Muhlemann *et al.* 1990]. The microcomputer-based prototype solution has since been successfully translated into a commercial product which is known as ELMS (EMM Lane Manufacturing Software) [ELMS 1990].

ELMS was developed as a set of integrated software templates which comprise the following "core" or principal production management functions [Muhlemann *et al.* 1991] :

"Core" Production Management functions

1. Production Planning
* Material Requirement Planning
2. Production Progressing
3. Materials Management
4. Costing

The ELMS software application has been developed based on the use of a proprietary relational database, namely DP4, and a fourth generation language, namely Datafit, to aid data manipulation, representation and access. It basically consists of a group of executable programs which operate on the database in order to realise the functionality of the "cores".

Further research work is currently being carried out at UBMC to enhance the basic functionality of ELMS to provide a resource scheduling capability [Halsall *et al.* 1993]. The work entails the development and incorporation of the required scheduling capability where its information needs could be satisfied and derived from the existing underlying database. However, due to the proprietary nature and lack of understanding of the database structure (in terms of the information entities represented and their dependency and interrelationship defined within the database), the task of identifying the relevant information necessary to support resource scheduling has proven to be very difficult and demanding. This problem is typical for such "as is" software systems, a property which can severely inhibit their future development.

Thus as an initial step towards facilitating ease of further development to incorporate resource scheduling capability in ELMS (and indeed future functional enhancements as required), work has been carried out at UBMC to restructure the ELMS database with reference to the generic information models proposed by the author in Section 4.2. As illustrated in Figure 4-3, this research has involved the following activities:

(i) Identification of specific information entities from the ELMS proprietary database which correspond closely to those represented in the generic information models (refer to Figure 4-4 for illustration).

(ii) Establishing a mapping between those closely associated information entities. Please refer to Figure 4-5 for illustration.

(iii) Populating with data the mapped information entities (contained in the generic information models) with relevant physical data from the existing ELMS database.

Based on this restructuring, the information entities required for resource scheduling (which are stored with reference to the generic information models) are listed in Table 4-5.

This case study clearly demonstrates the ability of the generic reference models (identified by the author in this research study) to

- provide clarity in the database schema where information entities and their attributes are clearly defined so that they may be well understood within the enterprise.

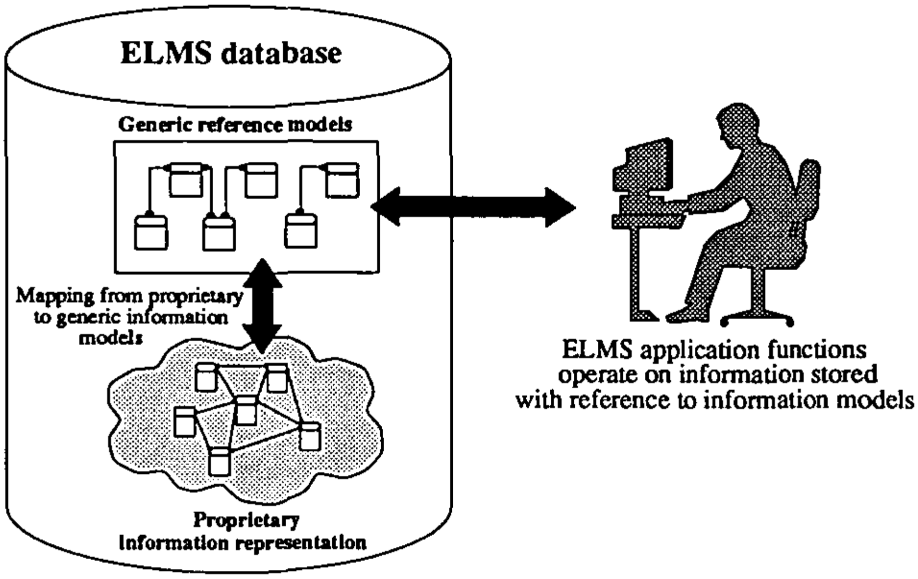

ELMS database
Generic reference models
Mapping from proprietary to generic information models
Proprietary information representation
ELMS application functions operate on information stored with reference to information models

Figure 4-3 : Restructure ELMS database with reference to generic information models

	Information models ⇦	Proprietary information
Parameters for resource scheduling - Part quantity required - Production due date - Suppliers lead times - Operation sequence - Work groups or cells - Stock level	Order Entry	Customer Order Work Order Usage
	Schedule	Work Order Item
	Process Plan	Operation Process Item Operation
	Resource	Item
	Part Master/BOM	BOM
	Manufacturing Facility	Process
	Manufacturing Cell	Work Group

Table 4-5 : Information requirement for resource scheduling in ELMS

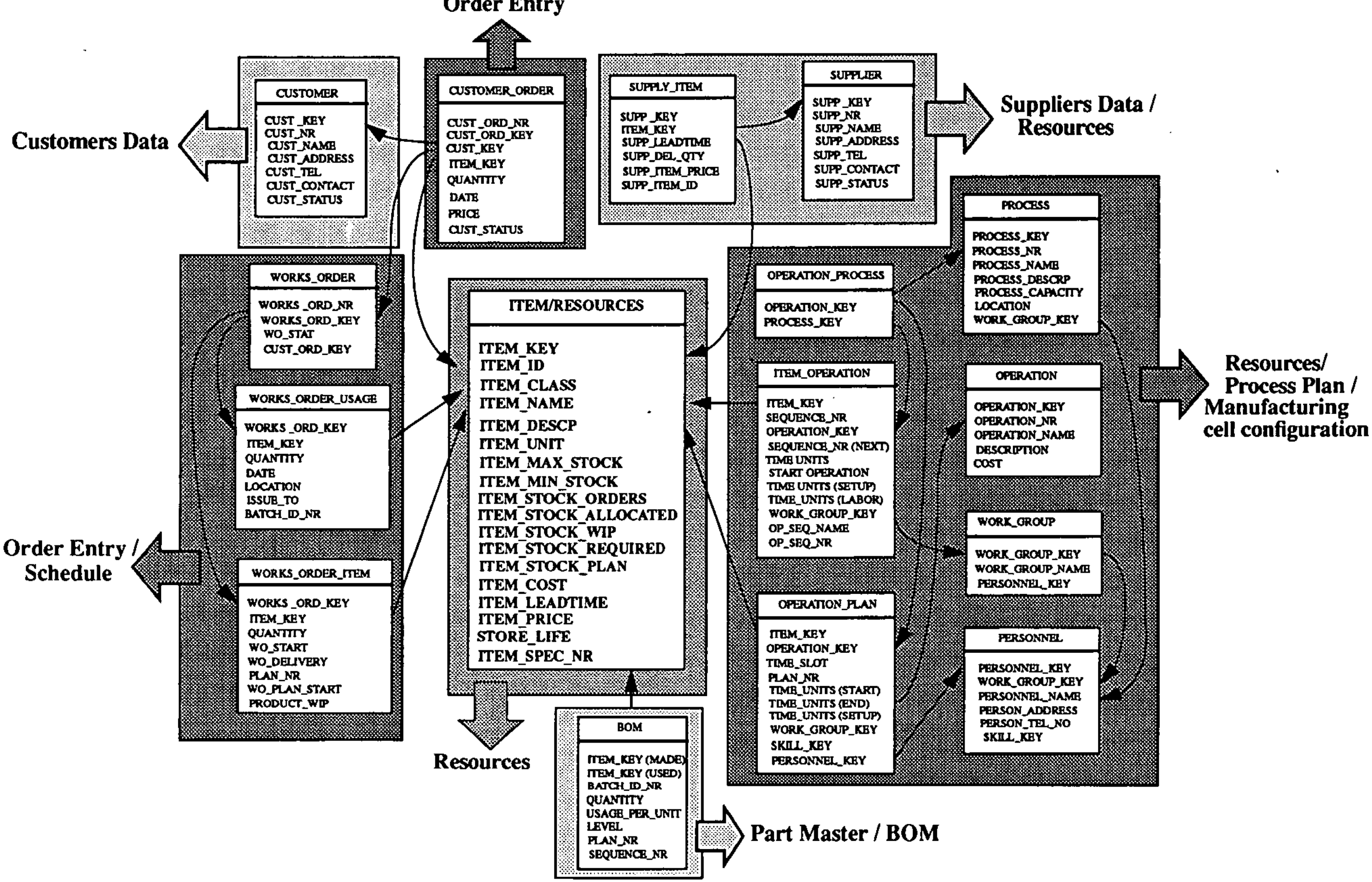


Figure 4-4 : Grouping of data represented in ELMS database with reference to information models

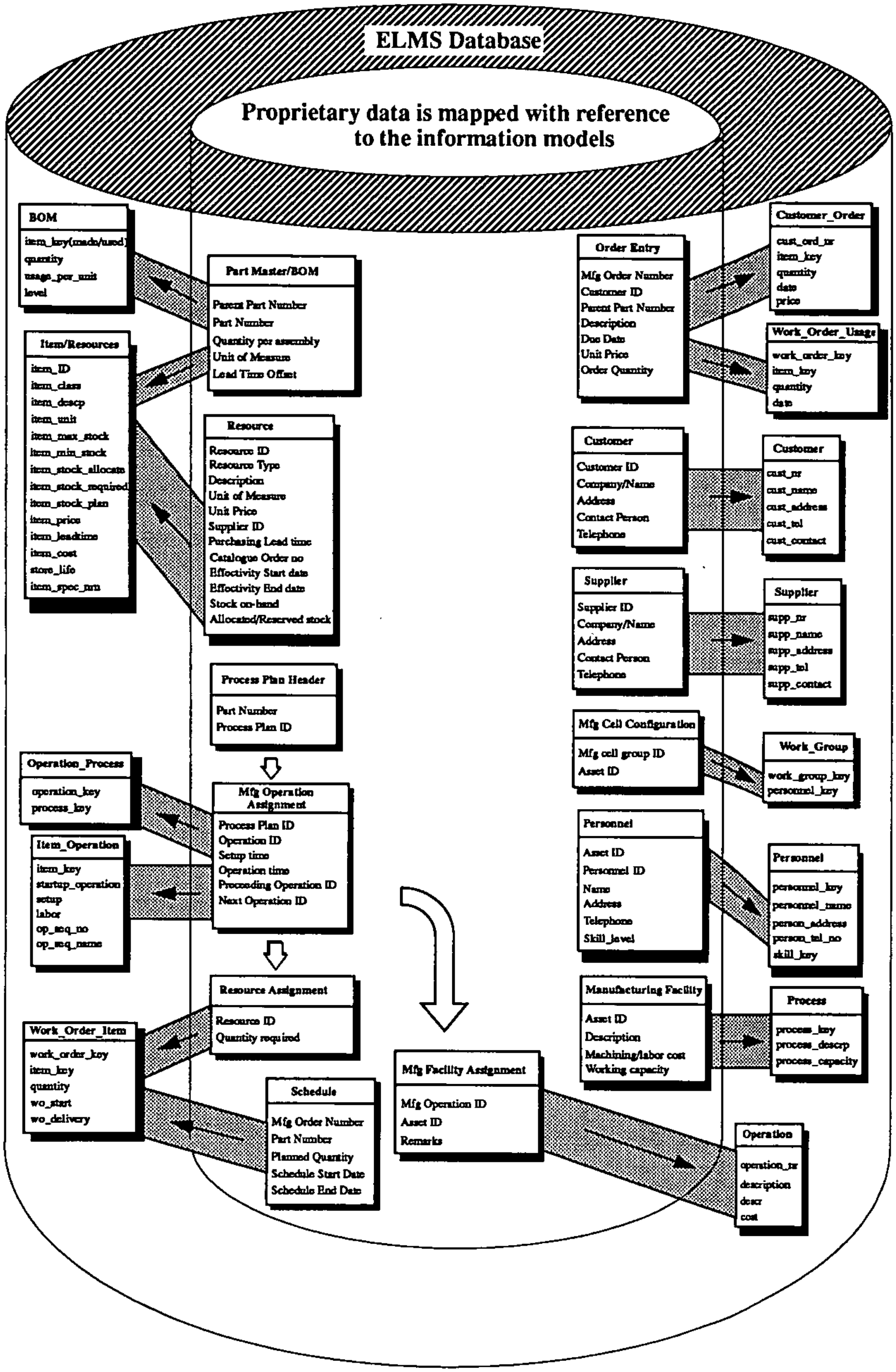


Figure 4-5 : Mapping between information entities

- increase the flexibility of the database so as to enable information required by the functions concerned to be made easily and quickly available (with flexible association between information entities).
- transform from a proprietary database schema to a more widely applicable and formally structured schema for which future change can be more readily supported.

One of the major benefits of the approach adopted is an enhancement of ELMS from a stand-alone "as is" CAPM software application to one which now has the potential to more readily interoperate with various other MCS applications. This is made possible because the information stored with reference to the generic information models in the underlying database constitutes information of common interest which is typically shared between various other MCS applications concerned (such as product design, cell control and process planning systems). Although the results of a single case study are reported here the author is confident that the information model and the method used to transform between proprietary and neutral representation could also be used to achieve enhancement to many CAPM software packages of similar structural design.

4.4 Design Criteria for a System-Wide Data Repository

As part of the information architecture proposed by the author, a new approach to consolidating MCS data is offered. The approach was conceived after having recognised the need for solutions in the following problem areas (as highlighted in section 2.2.2) :

(A) Lack of adherence to any standard architectural models of information, which undermines the semantic integrity of shared information, can result in invalid combinations of data (this as a consequence of incompatibility and inconsistency in data definition and format).

(B) Multi-database concurrency problems, related to database transactions and concurrency control and a means of consistently handling data access, transfer and presentation issues, for data which is normally fragmented and distributed across various databases. This includes

- independent and transparent data access, i.e. access without having to specify or know where the data is stored.
- support for multi-user access where appropriate in-built system mechanisms ensure that competing applications wishing to access data entries do not endanger the integrity of the data repositories.

The data repository comprises a directory of shared elements where common data definitions are recognised throughout the enterprise as global data elements; i.e. they have globally understood inter-relationships. *The system-wide data repository would serve as the focal point for access of shared data* and provides users and software applications with a consolidated view of information - one that is independent of physical media or data location; refer to Figure 4-6 for an illustration of the concepts involved here.

It should be noted that data which will always be unique to an application can remain in its own private local database, this to realise efficient processing. By restricting the number of information entities made globally available there will be a diminution in the amount of data exchanged, thereby reducing network traffic and bottlenecks relating to data access.

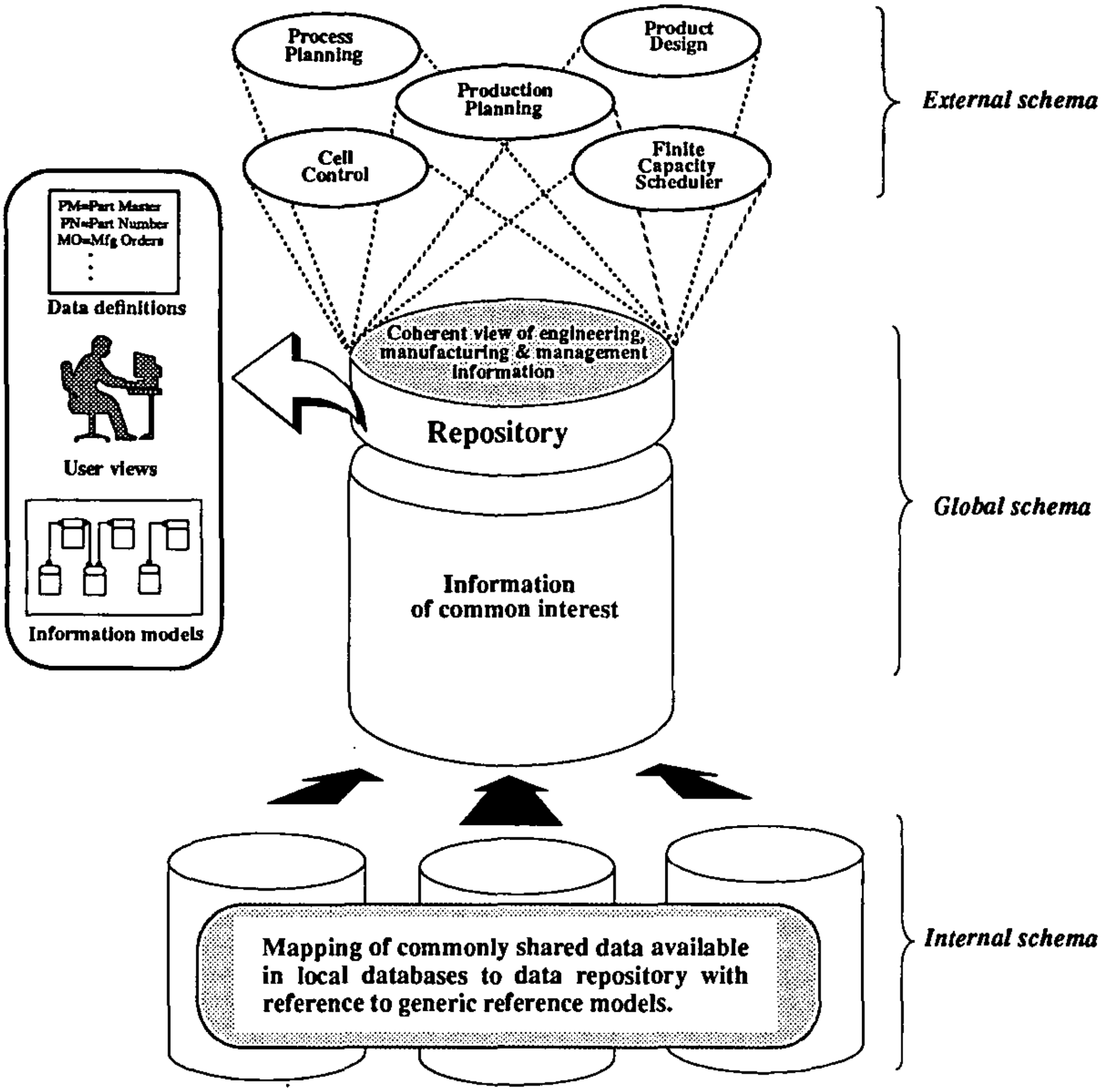

Process Planning
Product Design
Production Planning
Cell Control
Finite Capacity Scheduler
External schema
PM=Part Master
PN=Part Number
MO=Mfg Orders
Data definitions
User views
Information models
Coherent view of engineering, manufacturing & management information
Repository
Global schema
Information of common interest
Mapping of commonly shared data available in local databases to data repository with reference to generic reference models.
Internal schema

Figure 4-6 : System-wide data repository

The data unification mechanisms of the information architecture conceived by the author in conjunction with other MSI researchers unifies the use of a distributed set of data repositories which will be located over several computer nodes connected over a local area network.

As part of this research study, the following techniques were proposed and advanced to help overcome the problem specified in (A) :

- Apply the generic reference models identified and defined by the author (as described in section 4.2) to offer a degree of formalism in terms of recognising information of common concern over an extended domain.
- Adopt a suitably defined logical data model (to be further discussed in section 4.4.1) to represent information entities and attributes in a uniformed manner where these entities comprise information fragments within distributed data repositories. This can
  - alleviate problems of incompatibility and inconsistency in data definition and format;
  - facilitate required changes to the data model and physical data.

The CIM-BIOSYS IIS was utilised in this study to address some of the inherent problems referred to in (B). It provides software applications with structured access to common integration services for communication and management of data stored in distributed data respositories (see section 3.4.1 for details). Database 'drivers' have been developed by other MSI researchers [Leech 1993] to enable the following capabilities to be offered via the IIS :

- connection and direct communication with a number of proprietary databases (in a manner which is independent of the physical location of the information fragments involved); and
- a means of accessing information held in the databases.

However, the previously available database 'drivers' (created by MSI researchers) were originally designed to provide low level services where the communication protocol adopted by the 'drivers' can only handle packets of data of limited size. Thus in their original form the available database 'drivers' were only capable of providing support for simple database queries with limited information access. As a result, they were found to be relatively ineffective in supporting data-intensive activities which require large amounts of data to be accessed, transferred and processed. Thus *as part of this research study, it was necessary for the author to further develop the capabilities of database 'drivers'* (as discussed in section 4.4.4) so that they incorporate an enhanced SQL capability which offers extended and more flexible functions that can satisfy varying needs ranging from file transfer to complex database queries.

4.4.1 A Logical Database Model of the Data Repository

A relational data model has been used as an underlying structure for accessing information fragments from the distributed data repository. In a relational model entities, relationships and attributes are represented in the form of two-dimensional tables known as relations. Records are assimilated into the rows of the table and each set of attributes forms a column. *In a relational database entities are stored totally independently*; i.e. the existence of a relation is not dependent on any other relation. Logical associations among the stored data are exploited through relational operations such as *select, project and join* which can be used to create new tables. Any number of operators and relations can be combined in a 'relational expression' and used to answer almost any query. *The entities, attributes and relationships of the conceptual data model can often be modelled directly as relations in a relational database model* [Martin 1980].

The use of the relational model rather than hierarchical or network models has been claimed to demand less of a compromise when representing and transforming real-world data relationships [Wilkinson and Winterflood 1987, Date 1986, Martin 1980]. This claim is because of the following :

(i) In many real life situations, relationships cannot naturally be represented by a hierarchical model (where normally one-to-many segment types are used to represent successive levels in the hierarchy, thereby relating entities to one another). It is not easy, for example, to directly represent relationships among segment types at the same hierarchical level nor is it possible, without introducing data duplication, to represent many-to-many relationships between entities [Martin 1980].

(ii) Network structures offer greater scope in representing data relationships when compared with hierarchical structures, albeit at the expense of simplicity. This is certainly true of their underlying physical storage structure. The need to transform many-to-many relationships by the construction of a network model results in the need to make more or less irreversible decisions about the nature of relationships between entities when the data model is designed [Taylor and Frank 1976]. It should be noted that the network model, whilst permitting means of representing many-to-many relationships without introducing duplication of record occurrences, does make retrieval of data a laborious process [Olle 1978].

Please refer to Appendix I for further details on the relational, hierarchical and network data models.

The importance of the relational model is widely acknowledged and the development of relational database management systems (RDBMS) is progressing rapidly [Golberg 1993, Beeritbart 1993, Wilkinson and Winterflood 1987]. Potentially, the relational data model offers the following advantages :

- ***Ease of use.*** Visualisation and clarity of data, which are represented in two-dimensional tables, is better for both programming and non-programming users.
- ***Simplicity.*** Here all data is viewed in the form of relations (tables), thereby allowing easy data manipulation and query via SQL (Structured Query Language).
- ***Flexibility and relatability.*** With relational operations a standardised and effective way of decomposing and recomposing relations [Rusinkiewicz and Czejdo 1987] is provided. This approach enables the incremental building of larger systems module by module.
- ***Security.*** Security controls can be easily implemented where security authorization will relate to relations to protect company sensitive attributes.
- ***Data independence.*** There will be a need for most databases to grow by adding new attributes and new relations and also for data to be used in new ways. The relational model supports dynamic reorganisation (i.e. extension and modification to the structure) of the database without affecting existing applications [Maude and Willis 1991]. This is important because of the excessive and growing costs of maintaining the software applications of an enterprise and its data from the disruptive effects of database growth.
- ***Data manipulation language (DML).*** One of the strengths of the relational model is that it is generally supported by high level non-procedural, set-oriented languages, such as 4GLs (fourth generation languages), to enable flexible access, management and presentation of data stored in the database [Martin 1980].

As illustrated in Figure 4-6, within this study the "three-schema" architecture [Hodgson 1993, Hodgson *et al.* 1988, Date 1986] was adopted in order to map from the logically integrated relational model to the physically distributed databases in which the actual data is stored. The three schema approach requires definition of the following information schema :

- **Internal schema**
  This represents the physical organisation and storage of the information.

- **Global (or Conceptual) schema**
  This represents a composite view of a common pool of shared data. The objective is to provide a consistent definition for meanings and inter-relationships between data entities in order to aid information management.

- **External schema (or Local viewpoint)**
  This describes the use of information, i.e. the information required by a user or an application. Objects in the external schema are automatically mapped onto information attributes in the internal schema via reference to the global schema.

In this study *the shared elements represented by the global schema correspond to information entities and attributes which are defined and specified in the generic reference models* (as described in section 4.2). Hence with the three schema approach all commonly shared physical data which is fragmented across the various local databases of the data repository can be flexibly mapped and associated through established object relations. Thus each software application only requires one interface to the global schema, thereby enabling access to a common pool of information. Similarly, each local data source only requires one interface. Changes are required to the global schema if new data entities are provided or old entities are removed. In the event of a physical restructuring of the database, reorganisation only affects the internal schema and a single mapping between the global schema and the internal schema. Mappings between the external schema and the global schema remain unaffected.

Hence the three schema approach provides a greater degree of data independence when compared with two or single schema approach where information is typically embedded causing them to be exclusive in nature with regard to the functions concerned, thereby making it rather difficult to separate and access them. Thus the use of a three schema database architecture leads to improved flexibility where changes can be made without the need to modify applications. Hence individual local databases can retain their autonomy, with a focus on serving their existing customer set.

4.4.2 The Future Development of Relational Database Management Systems

It is the intention of the author at this stage to provide some insight into future development of relational database management systems (RDBMS). This provides a justification for choosing the relational model as a viable long term basis for manufacturing systems integration which can be expected to cope with advances in application requirements and database technology.

Traditionally, RDBMSs have lacked adequate data types to fully represent engineering data [Jain *et al.* 1992, Buchman 1984, Koriba 1983]. In a typical manufacturing company, engineering data will include CAD/CAM generated product models, part drawings and tool paths (used for generation of CNC programs). This engineering data is highly complex, where for example, relationships between the lines and angles of their vector graphics must be effectively represented and preserved. Within the literature, it is strongly advocated that such data be best handled and supported by some form of object orientation [Chaudhri and Revell 1994, Chaudhri 1993, Codd 1992, Maier 1989].

Progress is being made within the database research community towards extended RDBMS so as to provide increased functionality and support for object concepts. Examples include Intelligent SQL [Khoshafian *et al.* 1990], Objects in SQL [Beech and Ozbutun 1990], and current efforts of the ANSI X3H2 committee (SQL3) [Himes 1993, Hayes 1992]. Indeed major RDBMS manufacturers such as ORACLE, Sybase, Informix, Borland and Ingres have begun to add extensions to their products [Golberg 1993, Computing 1991], which can handle simple forms of object orientation, this to support complex and user defined data types which include :

- provision of large data fields that can store binary data;
- inclusion of stored procedures and triggers which allow for the storage of data along with programs and procedures that apply to them [Golberg 1993].

The reader is directed to one of the references given for details of the underlying concepts of the extended relational model. Hence choice of relational technology should provide a foundation for the future adoption of developed and enhanced forms which can also benefit from its inherent simplicity (which has been recognised as one of the great strengths of the relational model) [Chaudhri 1993]. Within the current scope of this research study, only non-engineering data, which is generally alphanumeric in nature and can be represented by the generic reference models identified and described in section 4.2, will be considered. However, in the long term there exists the prospect of managing both engineering and non-engineering data as object-oriented relational database technology matures and becomes readily available. Such requirements have been identified and considered in specifying the conceptual solutions proposed in this research study (see section 5.3 on management of engineering resources).

4.4.3 The Need for Database 'Drivers'

Depending on the class of user interacting with a database management system, two access approaches can be adopted for relational databases [Davis and Olson 1987] as illustrated in Figure 4-7 :

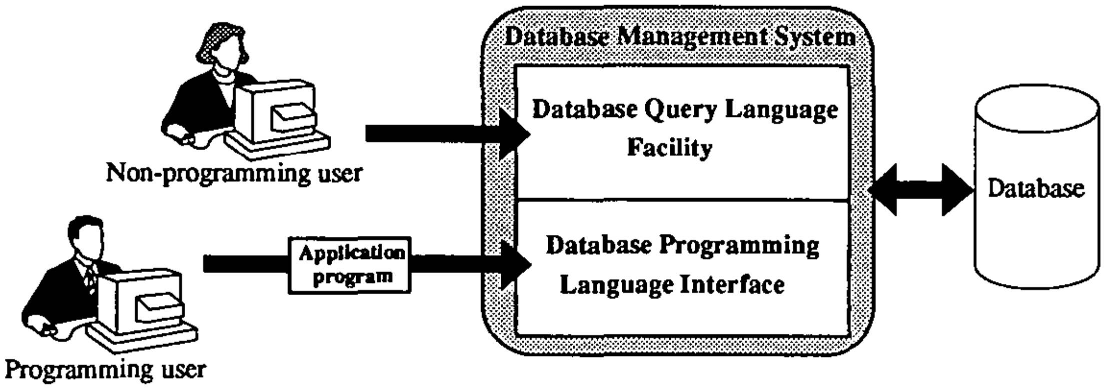

Database Management System
Database Query Language Facility
Database Programming Language Interface
Database
Non-programming user
Programming user
Application program

Figure 4-7 : Approach for database access

(I) Database Query Language Facility

Relational Database Management Systems (RDBMS) essentially adopt a table based organisation of data. The user must establish links from table to table, thereby making tables behave temporarily as relations. This can be done through the provision of fourth generation database query languages such as SQL (Structured Query Language). SQL in various forms is offered by different database manufacturers, for example : Query DL/1 for DL/1 (IBM); SQL for DB2 (IBM); SQL for ORACLE (Oracle); NATURAL for ADABAS (Siemens); SQL for Ingres (Ingres); and so on. Via the help of a basic structure, based on the following three key words, SQL allows complex queries to be made :

SELECT Fields (columns) to be displayed;

FROM Relations (files) containing the fields;

WHERE Conditions (giving the selection criteria).

The use of SQL is appropriate for non-programming users as it enables independent access to information necessary for the particular area of application concerned.

(II) Database Programming Language Interface

This approach is well suited for the programming user. Special database interface instructions (i.e. embedded SQL), which are normally offered as a set of readily available program functions, are incorporated into the application program to enable access to the database through the database management system. However, when using this approach, if changes to data access, manipulation and presentation are required, it would require a change in the application program(s).

Despite the widespread commercial and industrial use of SQL two major technical barriers currently exist with respect to data access across multi-vendor RDBMS [Krishnamurthy *et al.* 1991]. These two barriers are considered below :

(A) SQL language differences.

In reality each vendor has their own SQL "dialect" where each is peculiar to the RDBMS it supports. It has been estimated that approximately 80% of the SQL syntax and semantics is identical between dialects [Himes 1993]. However, the remainder includes subtle differences which make it extremely difficult to convert from one SQL dialect to another.

(B) Differences in message formats and communication protocol.

In practice different RDBMS vendors have targeted the use of their products at different hardware platforms and operating systems. As a result there exist distinct differences in the message formats and communication protocols of the different RDBMS which can further accentuate the heterogeneity of integrated systems. This undoubtedly causes interconnection problems between multi-vendor RDBMS.

A novel approach to accomplishing access to a distributed and heterogeneous set of data repositories, which has been used and is reported here, has been conceived by MSI researchers and advanced by the author in this research study by providing mechanisms in the form of specially developed database 'drivers' based upon SQL. As illustrated in Figure 4-8, the database 'driver' brings the database system up to a level of conformance which allows it to utilise the integration services of an IIS. The 'driver' is required to operate at the interface between the IIS and the database system. It provides a set of services to link the IIS and database system; it handles the IIS service requests and the inherent complexities of the database system concerned to access the information required.

In the implementation the CIM-BIOSYS IIS is utilised to provide distributed software applications with structured access, via common integration services for communication and management of data, to fragments of information stored in distributed data respositories. Here 'drivers' are generic in nature and are configurable, thereby enabling a degree of tailoring to account for the idiosyncrasies of the relational database management system to which they interface. Whilst efforts in SQL standardisation are progressing, which will ultimately make it easier to connect RDBMS from different vendors [Himes 1993, White 1993, Hayes 1992, Perkovic 1991, Van der Lans 1989], in the interim the *database 'driver' provides a migratory path towards more open systems which enable wider information access.* Thus the 'driver' acts as a standard interface to a relational database where its underlying services can be utilised by any MCS functions requiring access to the database via the IIS. To-date a number of database 'drivers' have been developed by MSI researchers to enable information

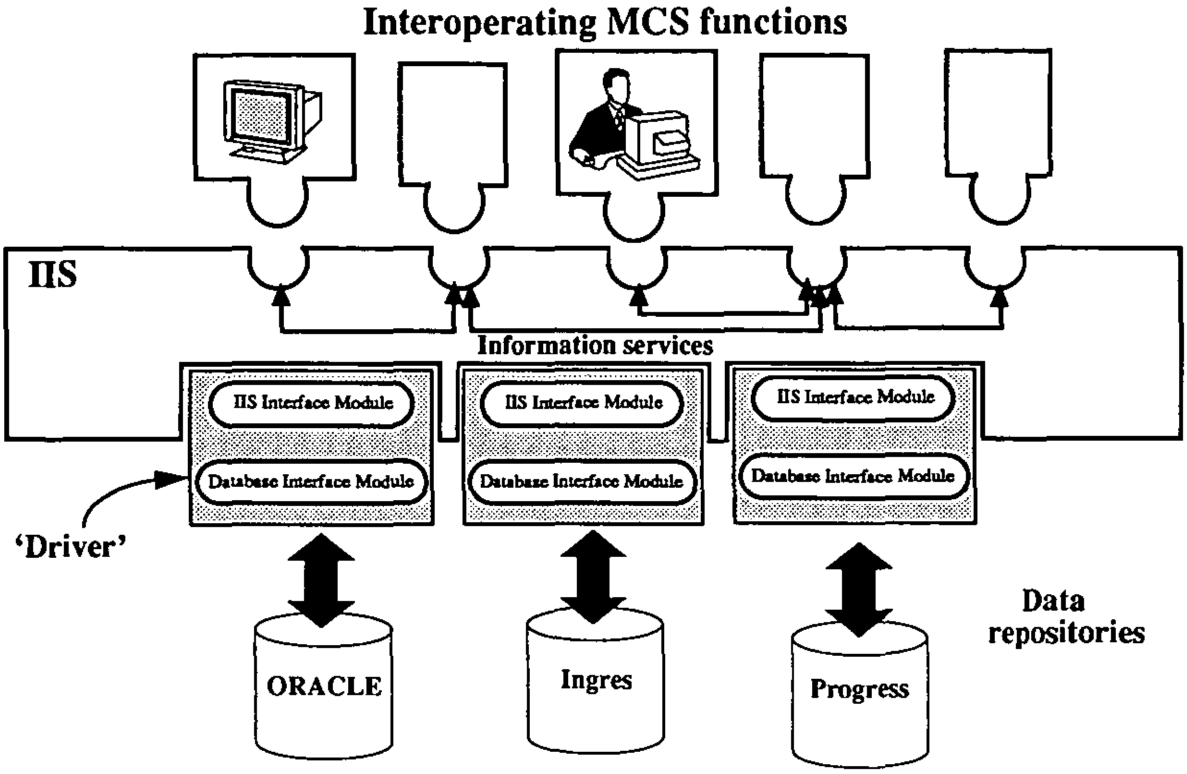


Figure 4-8 : Database 'driver' interfaces to data repository and IIS

access from a number of commercial RDBMSs [Leech 1993]. These include ORACLE version 6.0 [ORACLE 1992], Progress version 6.2 [Progress 1990], and Ingres version 6.4 [Ingres 1991]. All three databases provide an SQL user interface as well as some form of embedded SQL capabilities, and various forms of higher level 4GL programming language.

It should be stressed at this juncture that *the database 'driver' developed in this research study is an improvement on an earlier version developed by other MSI researchers* [Leech 1993]. In the earlier version, use of the 'driver' was restricted as it was capable of only handling fixed maximum length packets of data (limited up to 1500 characters only). This limitation was primarily attributed to the communication protocol adopted which involved UNIX based sockets responsible for handling and transmitting data packets over the local area network (this being Ethernet for the systems implemented during this study). Thus in its original form the 'driver' was only capable of supporting simple database queries. This limited information access involved laborious and time consuming overhead processing in terms of

- fragmenting the required data into the necessary packet size;
- handling the sequential transfer of these data packets; and
- reassembling the data from the data packets transferred.

As a result the earlier version of database 'driver' proved to be ineffective and inefficient in supporting data intensive activities which normally require large amounts of data to be accessed, transferred and processed.

4.4.4 Development and Enhancement of Database 'Driver'

The 'driver' developed by the author in this research study is aimed at enabling consistent, reliable, transparent and open access of information stored in the database system concerned by the distributed software applications via the IIS. It is developed as a set of ANSI-C program functions. The 'driver' is structured in a modular manner and comprises the following (see Figure 4-8) :

- ***IIS Interface Module*** to provide an interface to the IIS to bring the database system serviced to a level of conformance which facilitates utilisation of the common integration services (i.e. information management and access) offered by the IIS. This is to enable transparent access to information (which is held in the distributed data repositories) from MCS functions via the IIS embedded communication protocol.
- ***Database Interface Module*** to provide an interface to specific database system, configured to translate between IIS and proprietary (i.e. specific to the database system concerned), message and data formats as well as SQL dialect requirements. Here the database programming language interface of a given RDBMS (i.e. embedded SQL) is used to access the database system. It incorporates standard SQL capabilities and in combination with the IIS the following services are provided :

  Data Definition:
  •CONNECT: Start a database session with the specified database.
  •DISCONNECT: End a database session.

  Retrieve Data:
  •SELECT: Retrieve rows that meet a given search condition.

  Data Manipulation:
  •INSERT: Insert a new row in a database table.
  •UPDATE: Modify rows that meet a given search condition.
  •DELETE: Delete rows in a database table specified by a search condition.

  Transaction Processing:
  •COMMIT: Make permanent all changes performed in the current transaction.
  •ROLLBACK: Undo the work done in the current transaction.

  The reader should refer to Appendix V for further details on the information services offered by the database 'driver'. Program listings for the relevant services are also included.

Due to the modular structure and generic feature of the 'driver', the task of creating 'drivers' for different database systems is simplified considerably. It is only required to modify the Database Interface Module to suit the specific requirements of the database system concerned.

For example, the ORACLE RDBMS 'driver' was created by taking the completed Ingres database 'driver' and replacing its Database Interface Module after modifying it to suit ORACLE RDBMS.

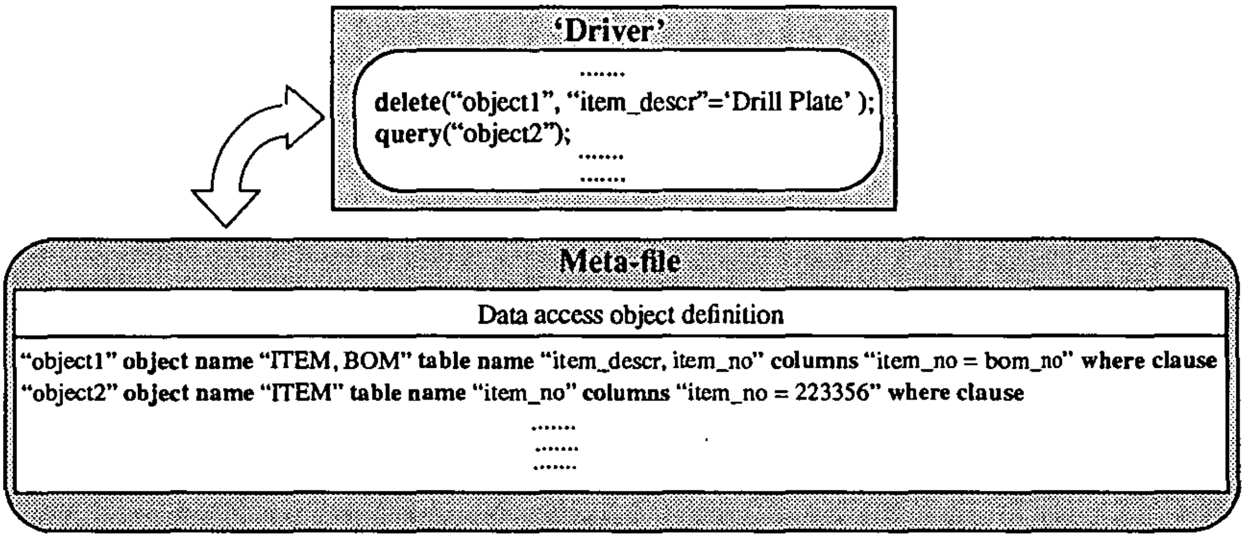
The operation of the 'driver' is initiated by the IIS to perform the set of tasks listed below :

- Establish connection to the database system concerned.
- Route information access requests received from MCS functions (via the IIS) to the database system. These requests are converted and conform to the SQL compliant services provided by the 'driver', i.e. SELECT, INSERT, UPDATE, and DELETE data objects.
- Receive the requested information from the database, re-format it and send it back to the MCS function which requested the information.
- Report and recover from errors which may occur.

During its operation the 'driver' references a set of meta-files which describe data access objects. A data access object is uniquely identified and consists of one or more of the database table names from which data is to be accessed, a list of field names and an optional search condition. The data access objects are predefined and can be altered easily in the meta-file (even during run-time if required) to suit specific information needs. This object oriented approach helps to avoid the tedium of modifying the database 'driver' whenever changes occur. Please refer overleaf for illustration of the use of data access objects by the database 'driver'. The notion of data access objects

- makes manipulation and query services simple, flexible, fast and easy to use, as it is not necessary to repeatedly reconstruct syntactically correct SQL statements from scratch every time a service is requested.
- simplifies and enables mapping between the data representation used in the physically distributed databases and that used in the logical integrated relational data model of the data repository. This in essence represents the link between the internal and global schema respectively in relation to the three schema architecture (as discussed in section 4.4.1). Please refer to Appendices IV and VI respectively for an illustration of the way in which this mapping was established between the generic reference models (described in section 4.2) and specific information models of two proprietary software packages used in this study, namely MCC [MCC 1989] and ELMS [ELMS 1990]. The program source code designed and developed by the author to populate the data repository with shared data from MCC is also included in Appendix VI for reference.

Illustration of the use of data access objects



'Driver'

```
.......
delete("object1", "item_descr"='Drill Plate' );
query("object2");
.......
.......
```

Meta-file

Data access object definition

"object1" **object name** "ITEM, BOM" **table name** "item_descr, item_no" **columns** "item_no = bom_no" **where clause**
"object2" **object name** "ITEM" **table name** "item_no" **columns** "item_no = 223356" **where clause**

....
....
....

- The 'driver' is invoked and operates on data access objects. In the example given, a data access object called **objectl** leads to the deletion of all data satisfying the search condition **"item_descr" = 'Drill Plate'**

- **objectl** is defined in the meta-file. It provides a composite description of a data access to the SQL tables ITEM and BOM which in turn involves attributes "item_descr" and "item_no" and the need to satisfy the search condition **"item_no = bom"**

Hence *the database 'driver' developed in this research study is an improvement on an earlier version developed by other MSI researchers.* It incorporates standard SQL capabilities and offers extended and more flexible functions to support data intensive activities which range from file transfer to complex database queries via the IIS.

4.5 Summary

In relation to the overall methodology conceived in this research study to enable interoperation of MCS components (as described in section 3.5), the work described in this chapter has a direct bearing on following three sub-methods :

- **MCS specification**
  A set of widely applicable generic reference models representing prime information of common concern among MCS components has been identified and described in order to help structure and facilitate MCS design.
- **Use of an IIS**
  The CIM-BIOSYS IIS, which assumes responsibility for maintaining knowledge of integration details, is used to simplify problems of realising interconnection between MCS components. It resolves differences in the physical system relating to heterogeneity, distribution and data fragmentation.
- **Interfaces to physical resources**
  An SQL based database 'driver' has been specially developed to effectively support data intensive activities. The 'driver' enables the following :
  - brings MCS components to a level of conformance which enables interoperation over the CIM-BIOSYS IIS, and hence allows practical application with regard to existing custom designed database components.
  - flexibly maps software applications onto system resources, i.e. achieves a mapping between physically distributed databases and the logically integrated relational data repository which corresponds to the ANSI SPARC "three schema" architecture.

Chapter 5

Integrating Infrastructure to underpin MCS Interoperation

5.1 Functional Interaction Requirements in MCS

The author recognises that the identification and specification of *generic reference models* (as described in section 4.2), to enable a bonding of MCS components through shared information, *is merely an entry point towards achieving software interoperability* [Singh and Weston 1993]. In order that the benefits of software interoperability can be more fully realised, the next crucial step is to address problems of functional interaction between MCS components [Hars 1990]. In practice, this demands cooperation among interoperating software components to establish well defined communication channels which collectively promote and enhance intra-organisation interaction and co-ordination of activities [Singh and Weston 1994a]. Hence an inherent capability to control system behaviour is essential. Indeed this is one of the requirements (as stated in section 1.4) which need to be satisfied to enable software interoperability in an effective manner.

A primary objective of this research study is to provide a generalised and flexible way of enabling functional interaction between the components of integrated manufacturing systems. Hence a framework is required which formally structures and manages functional interaction between MCS components. In this way a group of normally autonomous MCS components can function as a co-ordinated whole, with means of maintaining discipline among them [Singh and Weston 1994a]. The following requirements are identified as being necessary to enable and support functional interaction :

(I) Establish formal association between MCS functions and their required information. Typically in a manufacturing company, the sequence of activities necessary to perform part manufacture in progressive stages, relies heavily on their functional dependencies and on information needs between the different stages. Generally, the association between functions and their required information is not formally and clearly defined, particularly across different functional domains, and much depends on the users to ensure control and co-ordination of the system [Lars 1990]. Consequently, this contributes to significant delays in transactions as well as misunderstandings and conflicts. Hence the existence of a formal association, established between MCS functions and their required information, is viewed to be highly beneficial in terms of providing means to structure and govern system behaviour (during run-time), where associated preceeding and succeeding activities and consideration for their shared information needs (as defined by the generic reference models in section 4.2) have to be taken into full account.

In addition, any associations established between MCS functions and information must be configurable in nature. This is necessary to suit both specific and changing user functional and information needs.

(II) Capability for controlling and co-ordinating the sequence of MCS activities.

Consideration needs to be given to the availability of shared information necessary to satisfy and support each of the (run-time) activities carried out by an MCS. This needs to be done with close reference to associations established between MCS functions and their required information, as stipulated in (I). Mechanisms will be required to closely monitor the availability of such information (held in the data repository), and to initiate or trigger appropriate functional activities in need of that information for further processing.

(III) Means to effectively coalesce all interacting MCS components.

It is necessary to provide easy user access to associated functions which may be distributed across the local area network so that human centred tasks can be appropriately supported. Such an approach can provide users with a global perspective when conducting their specific tasks so as to ensure better and more informed decision-making.

(IV) Means to manage engineering data.

Normally in a manufacturing company, some engineering data, which includes part drawings, is frequently referenced to help determine resource requirements (for example toolings, fixtures, etc.), plan route for part manufacture, part inspection routines and so on. Controlled access to this engineering data can prove very useful and can effectively contribute globally towards more informed decision-making and planning. Thus it would certainly be helpful to have knowledge of the existence of such data and where it can be accessed or requested within the system. As RDBMSs which manage both engineering and non-engineering data are not currently available (as discussed in section 4.4.2), a means is, therefore, required to manage such useful engineering data where the user can at least be informed of its availability and physical location over the local area network.

5.2 Contemporary Solution to enable Functional Interaction

As indicated from the author's academic and product literature survey, there are very few solutions available which have been designed with the purpose of tying together a set of software applications into a coherent system (i.e. to facilitate functional interaction). SIM (Systems Integration Manager) [SIM 1993], which is a commercial software package from Manufacturing Systems Portfolio PLC (based in Windsor, UK), is one of the few contemporary solutions available. To help assess the capabilities of contemporary commercial solutions to the functional interaction problems, based on the author's working experience of SIM, its scope of functionality and level of effectiveness in facilitating functional interaction (with relation to the requirements identified in section 5.1) was explored. This case study was beneficial in terms of providing a general indication of the focus and usefulness of a class of contemporary solutions aimed at enabling and supporting functional interaction.

5.2.1 Overview of Systems Integration Manager

Messages are the core of the SIM system and interoperation between applications is achieved through message passing. SIM provides the following features listed in Table 5-1 [SIM 1993] :

Features	Description
MESSAGE PASSING	The routing of messages between applications.
MESSAGE HANDLING	The definition and validation of messages.
EVENTS AND ACTIONS	The specification of the occurrences of conditions such as the receipt of a given message and the action to be taken.
PROCESS CONTROL	The administration of applications.
FILE TRANSFER	The transfer of text files between computers.
STATISTICS	SIM maintains statistics on the frequency of service requests.
PRINTING	The printing of files.

Table 5-1 : Features of SIM

All these features are available as a set of ANSI-C program functions and need to be incorporated into newly developed and existing applications in order to enable an interface to SIM to be established, thereby enabling use of the facilities it provides to allow interoperation.

The behaviour of SIM is described, defined and governed through a set of configuration tables, as illustrated in Figure 5-1. In order to be valid and accepted by SIM, the configuration tables comprise the following :

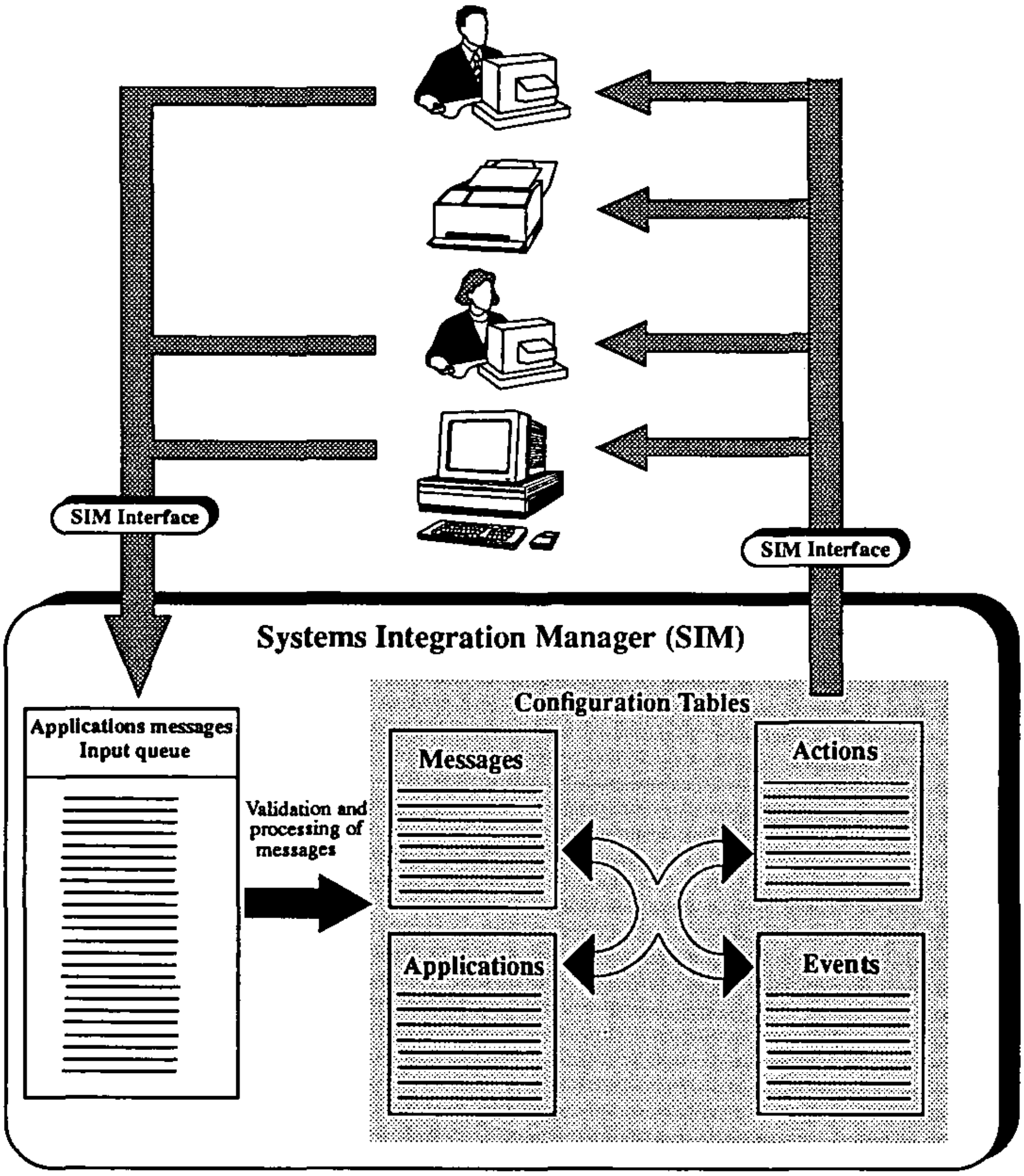

SIM Interface
SIM Interface
Systems Integration Manager (SIM)
Applications messages Input queue
Validation and processing of messages
Configuration Tables
Messages
Actions
Applications
Events

Figure 5-1 : Interoperation between applications enabled through SIM

- Registration of ***applications*** which includes assigning a unique identifier to each application and specifying its actual physical location for direct access.

Example

Event ID	21
Message Receipt	5
Action ID	10
File Transfer Request ID	260

- Predefinition of the structure of ***messages*** (which includes its type, length and assigned identifier) and a specification of its targeted destination.

Example

Action ID	10	
Application Loaded	12	◀— Application ID
Application Unloaded	16	
Application Terminated	14	

- Specification of ***"events"*** which can be described as some expected happening. For example, receipt of a message or the transfer of a file.

Example

Application ID	12	
Machine Address	tracey	◀— Logical name assigned to machine
Application Name	/usr/SIM/tools/apply1	◀— Application full path

- Definition of ***"actions"*** that should be performed when "event" happens.

Example

Message Type	ASCII
Message Text	Generate Schedule report
Message length	52
Destination Application ID	12
Message ID	5

As illustrated in Figure 5-1, during normal operation incoming messages from applications are received and queued by SIM. They will be validated and processed with reference to the configuration tables. In response to the input received, predefined "actions" and corresponding output messages will be activated and delivered respectively by SIM to trigger off targetted applications.

5.2.2 Focus and Limitations

In relation to the requirements identified for functional interaction (as described previously in section 5.1), the following highlights the characteristics of the contemporary solution studied :

- **Event driven to control and co-ordinate a sequence of activities**
  It is basically event driven and is loosely based on predefined message passing between applications. *It does not take into account information resources* required to drive and support the various activities within the system. Thus association between functions and information is not considered and availability of required information is not monitored when attempting to validate and control the sequence of activities. Rather, system behaviour control (during run-time) is achieved merely through management of messages.

- **No effective means of coalescing interacting applications**
  No front-end user interface capability is provided to coalesce interworking among the various interacting applications. Instead all applications need to directly incorporate special routines within themselves so that they can utilise the interaction facilities and services provided. Hence the *applications concerned are rigidly configured as the interaction knowledge they require will be embodied within each one of them.* Thus such solutions will face similar problems to "pair-wise" integration (as mentioned in section 2.2.1), where the *complexity of such systems will grow substantially (theoretically in a square law fashion) as the number of interconnected applications grows.* As a result, *it will be tedious and cumbersome to manage changes to any of the interacting application*s within the system, and it may not be practical to give due consideration to the full implication of changes to the system as a whole.

In view of the above mentioned limitations, the following requirements are evidently necessary to facilitate and support functional interaction in a more effective manner :

- ***Control and co-ordination of the sequence of activities*** (i.e. system behaviour during run-time) ***should be data driven.*** This is to ensure that the use of information resources, particularly those sharing common information, are considered during functional interaction. This is essential in order to properly and accurately validate and co-ordinated the sequence of activities in relation to the availability of information set in the context of realising more global goals.

- ***Simplify interconnection between interacting applications via an IIS***. This is to provide common integration services to all conforming applications, via use of common integration services provided by an infrastructure.

5.3 MCS Functional Interaction Management Module Subsystems

In order to meet the identified requirements for functional interaction, an MCS Functional Interaction Management Module (FIMM) was designed and developed as part of this research study. The MCS FIMM serves as an application enabler to help formally structure and facilitate MCS functional interaction in a controlled and deterministic manner [Singh and Weston 1994a, Welz 1993]. As illustrated in Figure 5-2, it comprises the following subsystems which collectively can be viewed as constituting high level integration mechanisms and tools of an IIS (as defined in section 3.4) :

- **Function-Information Association Table**
  This represents an association, formally established and clearly defined, between MCS functions and their required information. It is referenced (during run-time) to govern system behaviour. Software tools have been developed in this research study to establish and configure such association between functions and information held in the **Function-Information Association Table**; the purpose and construction of these tools will be considered further in the next chapter.

- **Functional Interaction Manager**
  This is responsible for controlling and co-ordinating a required sequence of MCS activities where mechanisms are provided to
  - closely monitor the availability of shared information (held in the data repository).
  - initiate or trigger appropriate functional activities in need of that information for further processing.

- **Engineering Resource Manager**
  This serves as an archive for engineering data which includes part drawings and NC programs necessary to support part manufacture.

- **FIMM Configurator**
  This serves as a tool to enable configuration of the MCS FIMM to suit specific user needs.

The MCS FIMM builds upon the low level, general purpose integration services and tools of the CIM-BIOSYS IIS [Weston 1993] which provide standard data inter-communication and information transfer facilities to interconnect MCS software components. Program listings for the MCS FIMM are included in Appendix VII.

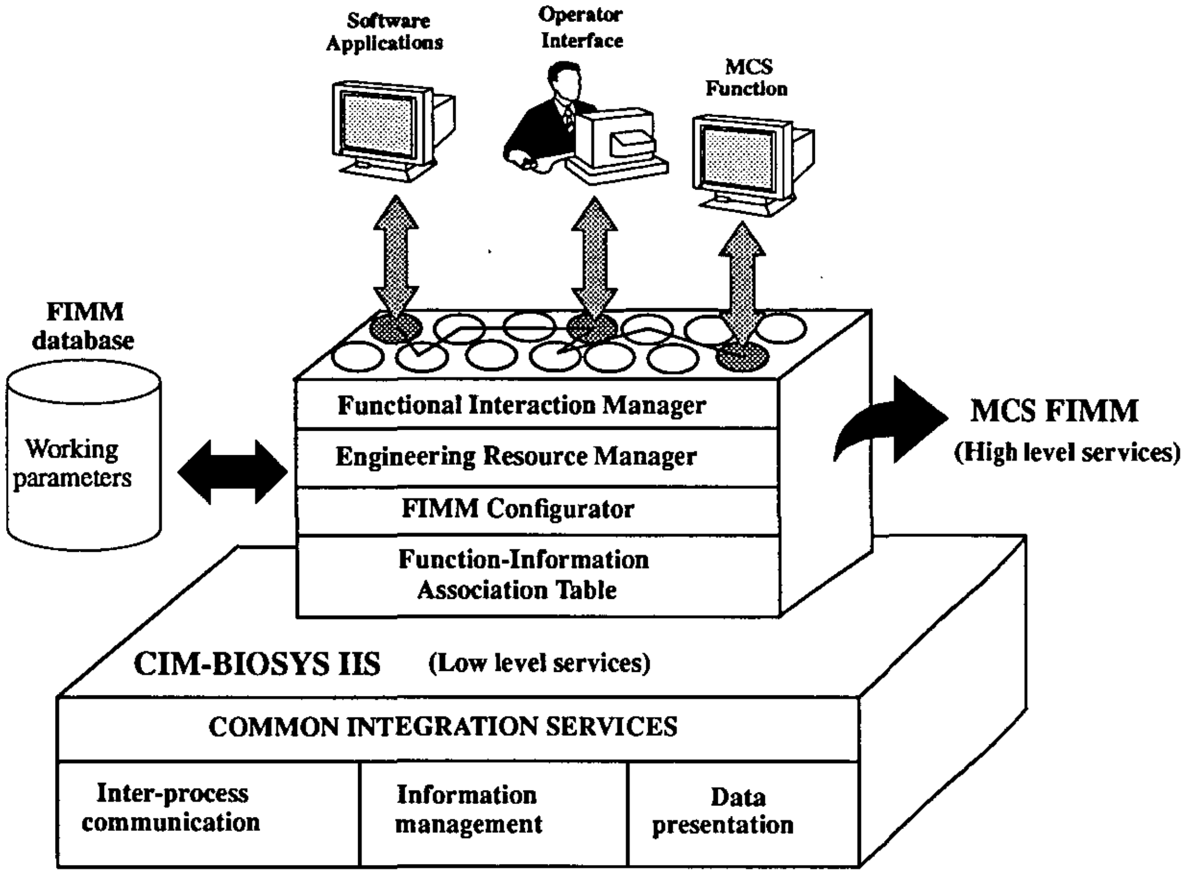


Figure 5-2 : Framework for MCS Interoperation

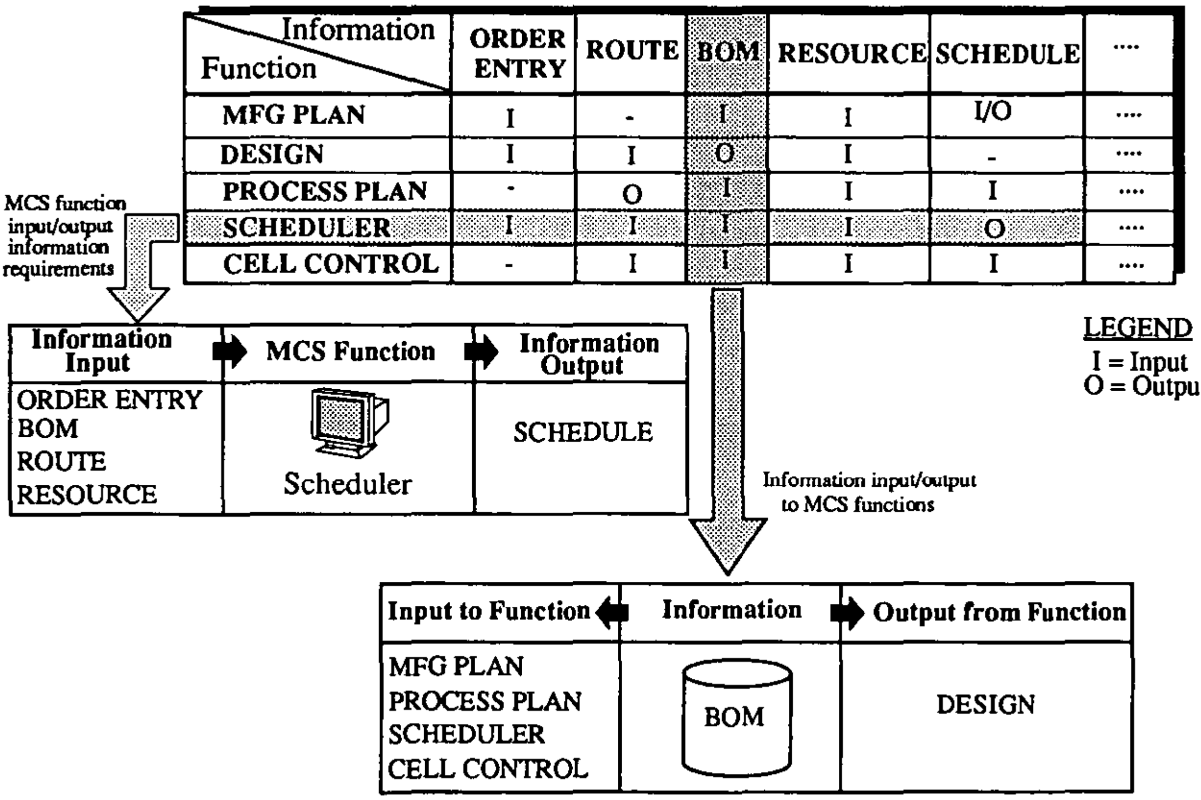
(I) Function-Information Association Table

As illustrated by Figure 5-3, the relational Function-Information Association Table identifies predefined relationships between entities of the information model and related MCS functions (i.e. it associates information entities to MCS functions designating them as either an input or output requirement or both). The information model, which encodes relationships between information entities, corresponds to the generic reference model (as identified and described in section 4.2). The Table will be referenced (during run-time) in a way which governs system behaviour in a controlled and co-ordinated manner, i.e. in accordance with predefined relationships established between the information entities and MCS functions. Use of this table driven approach offers the following benefits :

- information input/output requirement analysis with regard to the MCS functions; and
- information association, traceability and accountability to MCS functions.

This table can also serve as an intermediary storage and representation facility which can be populated with data generated from design models and information. Details of a methodology and software tools which have been specially developed and used to enact information and function models in this way are included in later chapters. This approach was conceived in order to establish and configure necessary run-time associations, as identified and represented by the Function-Information Association Table, in a highly flexible manner.

Function-Information Association Table



Information / Function	ORDER ENTRY	ROUTE	BOM	RESOURCE	SCHEDULE	....
MFG PLAN	I	-	I	I	I/O	....
DESIGN	I	I	O	I	-	....
PROCESS PLAN	-	O	I	I	I	....
SCHEDULER	I	I	I	I	O	....
CELL CONTROL	-	I	I	I	I	....

MCS function input/output information requirements

Information Input	MCS Function	Information Output
ORDER ENTRY BOM ROUTE RESOURCE	Scheduler	SCHEDULE

Information input/output to MCS functions

Input to Function	Information	Output from Function
MFG PLAN PROCESS PLAN SCHEDULER CELL CONTROL	BOM	DESIGN

LEGEND
I = Input
O = Output

Figure 5-3 : Function-Information Association Table for MCS FIMM

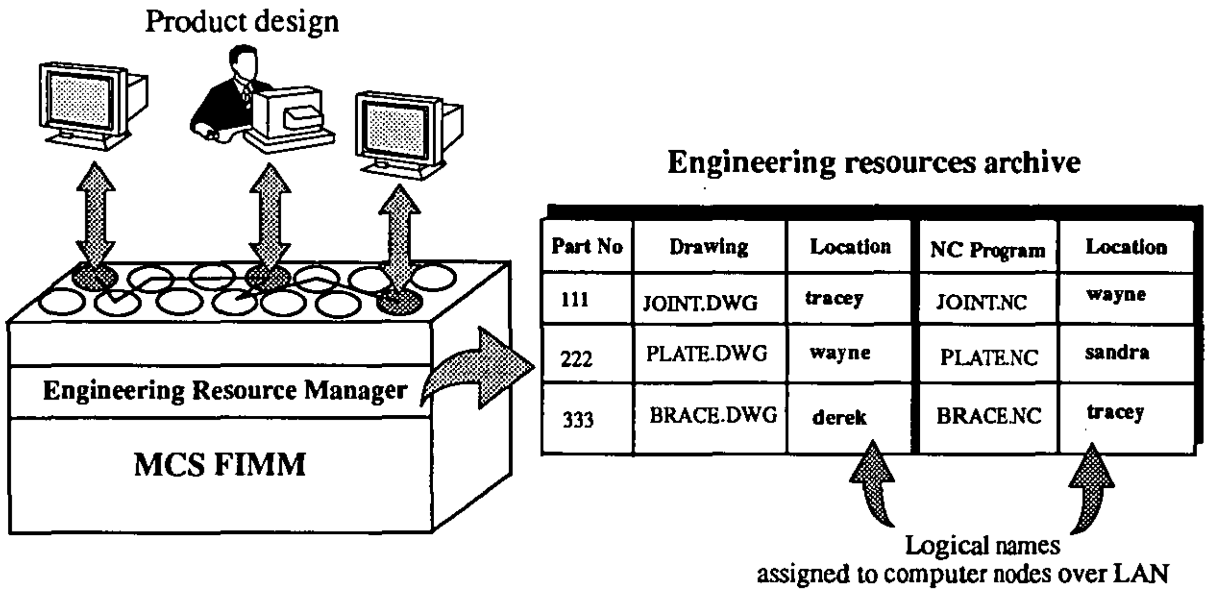
(II) Engineering Resource Manager

The Engineering Resource Manager offers a means of managing engineering data which includes part drawings and NC programs. It does not physically process this engineering data but effectively serves as an archive where these resources are registered in the FIMM database and cross referenced based on part number, as illustrated in Figure 5-4 below. Via the Engineering Resource Manager the user can be informed of availability and physical location of part drawings and NC programs, distributed across a number of computer systems.

The Engineering Resource Manager facilitates the following services :

- Registering the location of engineering resource.
- Search facility with reference to part number.
- Locate engineering resource with reference to the function responsible for managing it.

The author created Engineering Resource Manager software to meet the requirements highlighted above, program listings being provided in Appendix VII.


Product design
Engineering Resource Manager
MCS FIMM
Engineering resources archive
Logical names
assigned to computer nodes over LAN

Part No	Drawing	Location	NC Program	Location
111	JOINT.DWG	tracey	JOINT.NC	wayne
222	PLATE.DWG	wayne	PLATE.NC	sandra
333	BRACE.DWG	derek	BRACE.NC	tracey

Figure 5-4 : Engineering Resource Manager

(III) Functional Interaction Manager

The Functional Interaction Manager serves as the run-time driver of the MCS FIMM and is responsible for sequencing and co-ordinating activities of the various distributed MCS functions via status management and transaction control. Transactions which involve (i) initiation of interoperating functional activities, and (ii) request and exchange of shared data between interoperating MCS components are closely monitored with reference to the Function-Information Association Table (during run-time), this in order to validate

- that the required sequence of functional activities has been performed;
- appropriate association between MCS functions and their required information is maintained; and
- that the integrity of shared data in the system repository is maintained.

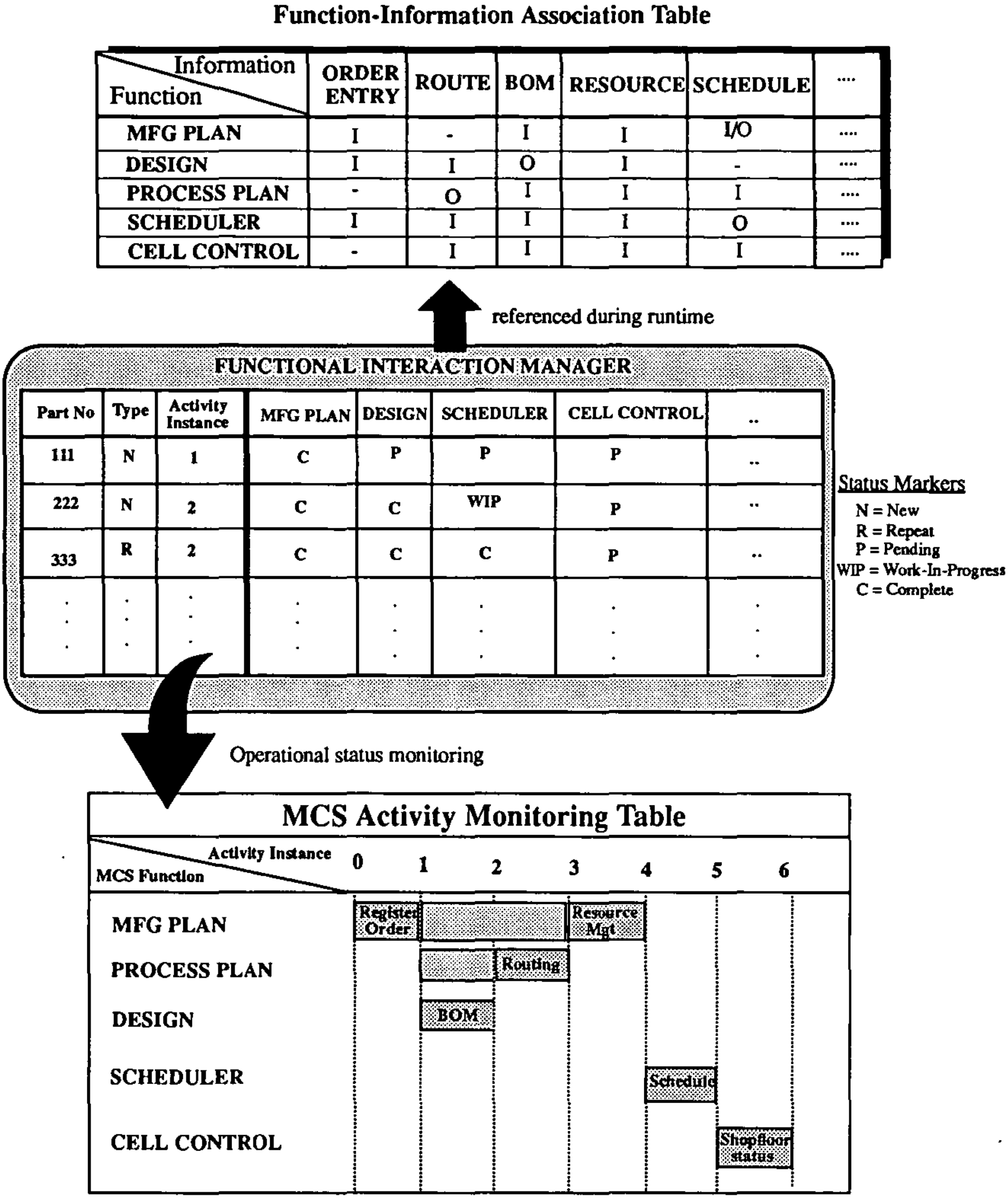
Status markers and triggers were implemented to provide a mechanism to control and co-ordinate functional interaction. Here, status markers are either assigned automatically or interactively by the operator (in response to certain system requests to alter them during normal operation), where they are subsequently used by the system to trigger or block further transactions between interoperating MCS components.

To enable concurrency, i.e. parallel operation of MCS components, the operational status of the threads of functionality embedded within each MCS function is monitored closely, for example : order registration and resource management activities which constitute the MCS manufacturing planning function. Status markers are also used to reflect changes of the activity instance, thereby effectively controlling and coordinating the start-up and shutdown of subsequent and dependent activities.

The Functional Interaction Manager facilitates the following services :

- Registering new orders received.
- Altering the processing status of MCS functions, i.e. to reflect whether work is in progress, completed or pending.
- Updating instance to reflect the processing phase at which the part is currently at.
- Search facility which references part number to indicate status and instance for the part(s) being processed.
- Job load indication for the MCS function concerned.

Please refer to Figure 5-5 for an illustration of the operational view of the Functional Interaction Manager and Appendix VII for further details on the services offered by Functional Interaction Manager. Program listings for the Functional Interaction Manager are also included in Appendix VII.



Function-Information Association Table

Information / Function	ORDER ENTRY	ROUTE	BOM	RESOURCE	SCHEDULE	....
MFG PLAN	I	-	I	I	I/O	....
DESIGN	I	I	O	I	-	....
PROCESS PLAN	-	O	I	I	I	....
SCHEDULER	I	I	I	I	O	....
CELL CONTROL	-	I	I	I	I	....

FUNCTIONAL INTERACTION MANAGER

Part No	Type	Activity Instance	MFG PLAN	DESIGN	SCHEDULER	CELL CONTROL	..
111	N	1	C	P	P	P	..
222	N	2	C	C	WIP	P	..
333	R	2	C	C	C	P	..
. .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Status Markers
N = New
R = Repeat
P = Pending
WIP = Work-In-Progress
C = Complete

MCS Activity Monitoring Table

Figure 5-5 : Overview of Functional Interaction Manager management of MCS activities

(IV) FIMM Configurator

A FIMM Configurator tool was conceived and implemented by the author to enable configuration of the MCS FIMM to suit specific user needs. It facilitates the addition, deletion, and display of MCS functions, which are to be managed by the FIMM, as well as enabling the attribution of data to entities in the information model (which corresponds to the reference model as described in section 4.2). In addition, it allows relations between the functions, information entities, engineering resources and activity instances to be altered quite easily through its editing services. Please refer to Annex VIII for further details on the services provided and program listings.

The FIMM Configurator is menu driven so as to be user friendly. Its functional structure and scope is illustrated in Figure 5-6.

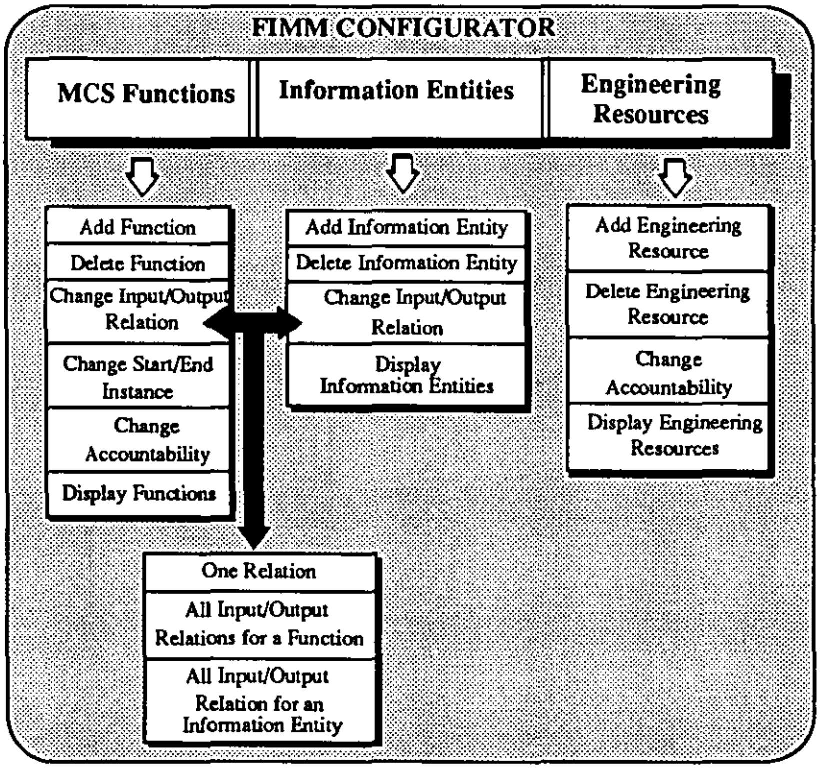

FIMM CONFIGURATOR
MCS Functions
Information Entities
Engineering Resources
Add Function
Delete Function
Change Input/Output Relation
Change Start/End Instance
Change Accountability
Display Functions
Add Information Entity
Delete Information Entity
Change Input/Output Relation
Display Information Entities
Add Engineering Resource
Delete Engineering Resource
Change Accountability
Display Engineering Resources
One Relation
All Input/Output Relations for a Function
All Input/Output Relation for an Information Entity

Figure 5-6 : Structure of FIMM Configurator

The configuration information describes the working parameters necessary to drive and control the MCS FIMM; during system run-time this information is referenced by the Functional Interaction Manager and the Engineering Resource Manager. Any alteration or reconfiguration carried out by the FIMM Configurator is done only when the MCS FIMM and all functional activities managed by it are inactive. This is to ensure consistency and integrity of the required configuration.

The following data which is relevant to the normal operation of MCS FIMM is stored locally in a relational database, namely the ORACLE RDBMS :

- attributes of the MCS functional activities and information entities to be managed;
- type and nature of relationship between MCS functional activities and information entities;
- operational status of MCS functional activities supporting part manufacture; and
- a description of the logical links to the engineering resources based on their designated call names.

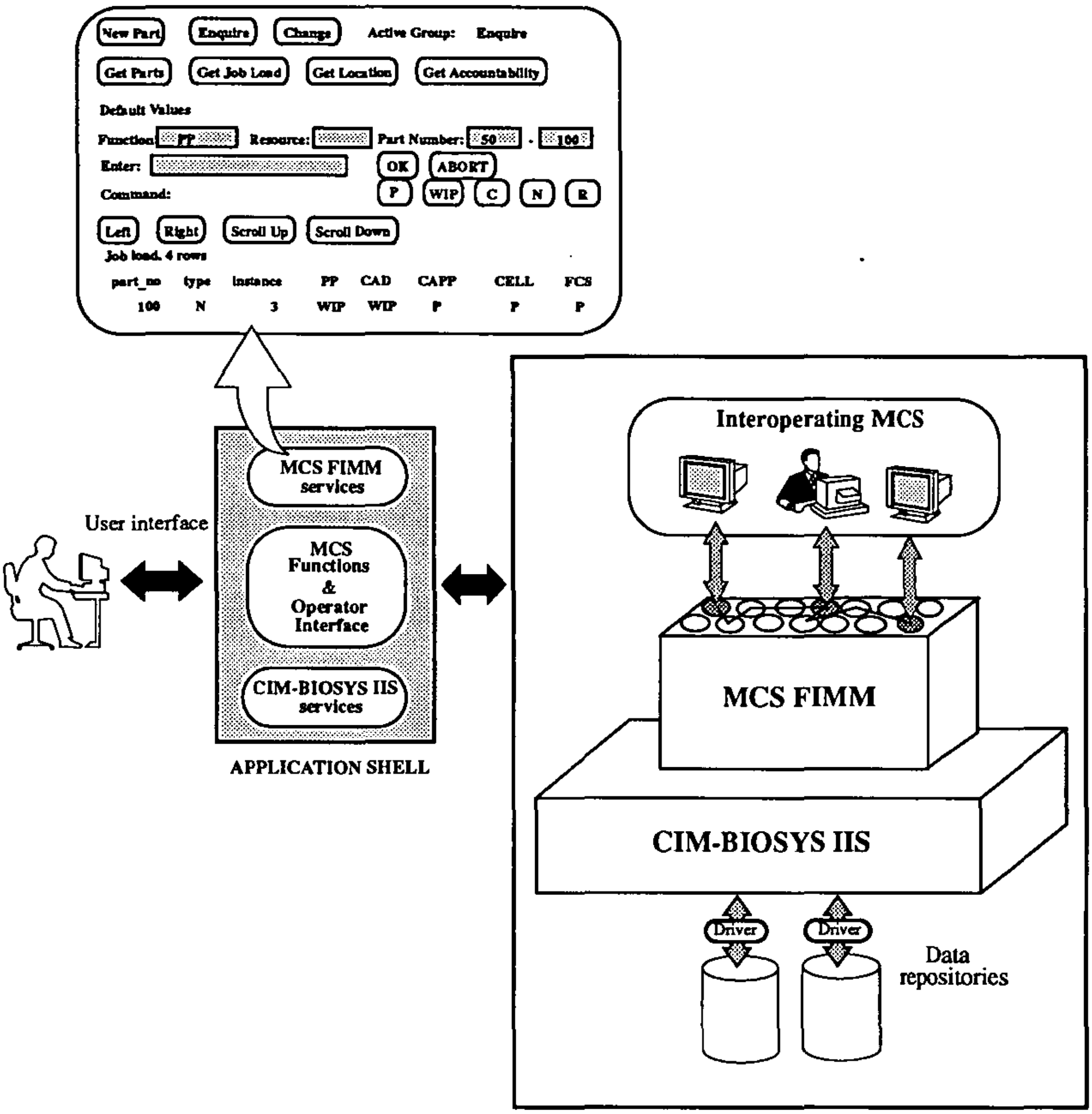
The reader should refer to Appendix IX for further details on the MCS FIMM database schema.

The FIMM Configurator also provides error recovery facilities to help troubleshoot and recover from possible error situations. Further details on the use of the FIMM Configurator tool is provided in the next chapter.

5.4 User Interface : Generic 'Application Shell'

As part of the working framework conceived by the author to enable functional interaction among MCS components, the need for a highly reconfigurable and hence widely applicable 'application shell' was perceived to be required. As illustrated in Figure 5-7, this 'application shell' effectively serves as a front-end user interface for MCS interoperation over the IIS, to provide easy access to associated MCS functions which are normally distributed within the complete MCS system. Through the 'application shell' operations such as start, stop, status update and queries (for both the MCS functions and MCS FIMM components) are performed in a standardised way irrespective of which MCS components are used. Common integration services, such as inter-process communication and information management, offered by the CIM-BIOSYS IIS are also accessed and managed through this standard interface.

The 'application shell' was designed and developed by the author in the form of a multi-window environment run under the UNIX operating system. The design included provision for data entry fields and specially configured buttons which serve as the front-end to perform and access the necessary MCS functions and services, as illustrated in Figure 5-7. The environment makes available all commonly required MCS functions, thereby allowing easy access to these functions for the following purposes :



Active Group: Enquire

Default Values

Function: PP Resource: Part Number: 50 - 100

Enter:

Command:

Job load. 4 rows

part_no	type	instance	PP	CAD	CAPP	CELL	FCS
100	N	3	WIP	WIP	P	P	P

Figure 5-7 : 'Application shell' to coalesce interoperating MCS components and integrate with MCS FIMM and CIM-BIOSYS IIS

- To provide users with an abstracted working viewpoint which supports them in an effective manner when conducting their specific tasks. This satisfies the need for a user interface capability which reflects inherent MCS-humàn interaction requirements (as stated earlier in section 1.4) so as to enable software interoperability in an effective manner. This 'standard' human-MCS interaction facility provides users with a global perspective which enables better and more informed decision-making in relation to performing their specific tasks.
- To make functional interaction management easier by effectively insulating the user from complexities and details involved in the underlying interoperation processes which will be taken care of by relevant services offered by the MCS FIMM and the CIM-BIOSYS IIS.

The reader should refer to Appendix X for further details on the human-MCS 'interface' offered through the generic 'application shell'.

5.4.1 Interface between MCS Functions and 'Application Shell'

In this study a proof-of-concept interface between MCS functions and the 'application shell' was implemented based on the use of UNIX pipes. A UNIX pipe basically makes the output of one program the input of another. Two pipes are created during normal operation. As illustrated in Figure 5-8, user inputs, which drive MCS functions, are emulated and redirected to one pipe. The output from the MCS functions is redirected to the second pipe to enable interpretation and visual display by the user via the 'application shell' display window. Program listings for the interface mechanisms implemented are included in Appendix XI.

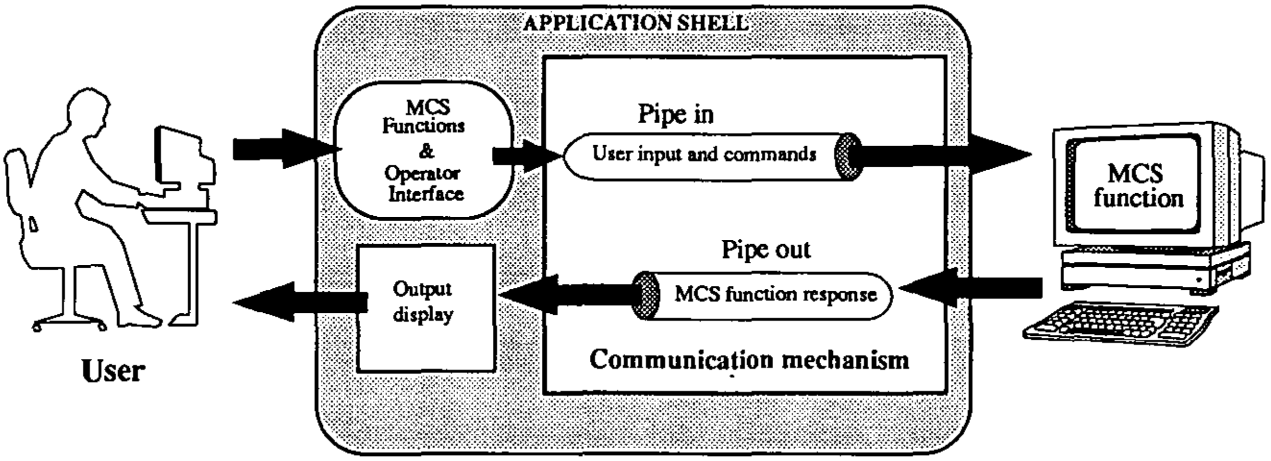


Figure 5-8 : Communication interface via 'application shell'

5.5 Enabling Distributed Functional Interaction Management

The services of the MCS FIMM are accessible as a set of ANSI-C program functions. If an MCS application runs on the host computer where the MCS FIMM database resides, the required services can be directly invoked as all required information is held locally. However, in order to support MCS functional interaction in a distributed environment, where the MCS functions are spread over several host computers connected via a local area network (LAN), remote versions of the services (as described previously in section 5.3) are necessary. These have been developed to allow any host computer connected to the network to have access to MCS FIMM services. Here the proof-of-concept implementation was based on communication mechanisms which use UNIX pipes to connect between the remote host computer on which the MCS component resides and the computer system on which the services of MCS FIMM reside. As illustrated in Figure 5-9, the communication mechanisms implemented enable the following capabilities :

- requests made by MCS applications to be relayed to the host computer where the services of the MCS FIMM reside for processing to occur.
- request made by MCS applications for MCS FIMM service(s) to be interpreted.
- MCS FIMM service(s) to be invoked as required.
- ability to relay responses or outputs from MCS FIMM services back to MCS applications requesting its service.

Program listings for the communication mechanisms implemented are provided in Appendix XI.

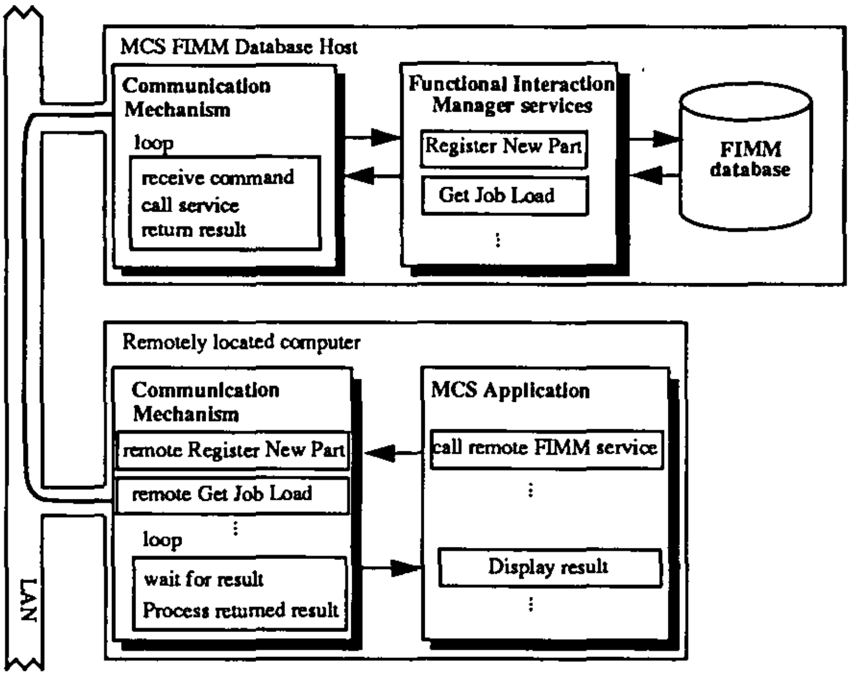


Figure 5-9 : Communication mechanism to support MCS FIMM in a distributed environment

5.6 Summary

The MCS FIMM functions provide a framework which formally structures functional interaction among MCS components. However it was conceived with the purpose of enabling the flexible integration of MCS components. Hence it is modular in construction and comprises the following sub-systems :

- **Function-Association Table**
- **Functional Interaction Manager**
- **Engineering Resource Manager**
- **FIMM Configurator**

The MCS FIMM was implemented in proof-of-concept form and developed to satisfy the requirements for functional interaction identified in section 5.1. Restated these requirements illustrated a need to :

- **Establish formal association between MCS functions and their required information.**

  These associations are represented in the form of a Function-Information Association Table which identifies predefined relationships between information entities (which correspond to the generic reference model described in section 4.2) and activities performed by MCS functions. This Table is referenced (during run-time) in a manner which governs system behaviour in a controlled and co-ordinated way. Here associated preceeding and succeeding activities and consideration for their need for shared information is taken into full account.

  The FIMM Configurator was conceived and developed to facilitate flexibility in a manageable fashion, where via its editing services, the MCS FIMM can be configured quite easily to suit specific user functional and information needs.

- **Establish a capability for controlling and co-ordinating the sequence of activities performed in a distributed MCS.**

  The Functional Interaction Manager is responsible for providing this capability. It incorporates mechanisms to

  - monitor the availability of shared information via its status management capability.
  - initiate or trigger appropriate functional activities which are in need of that information for further processing.

- **Establish means of effectively coalescing the interaction of MCS components via a 'standard' human interface.**

  A generic 'application shell' has been developed to enable human interaction with MCS components via MCS FIMM and CIM-BIOSYS IIS services. The 'application shell' effectively serves as a front-end user interface for MCS interoperation, where easy access to associated functions which are distributed throughout the MCS systems at different network nodes, is offered. This provides the user with a system-wide working viewpoint of the effect of conducting their specific tasks. The 'application shell' is responsible for drawing together various interacting MCS components in a manner which leads to synergy between them. As a result it promotes and enables cooperative as well as better and more informed decision-making across the enterprise.

- **Establish means of managing engineering data.**

  This was achieved via conceiving and developing an Engineering Resource Manager. This effectively serves as an archive for engineering data where users are informed of the availability and physical location of information entities which will be distributed as information fragments stored in different databases at a number of computer nodes connected to a LAN.

The MCS FIMM overcomes some of the limitations inherent in contemporary solutions (as discussed in section 5.2.2) by offering the following advantages when facilitating and supporting functional interaction :

- ***Control and co-ordination of the sequence of activities in a data-driven manner.***

  Here due consideration is given to the availability of shared information and the need by the functions concerned in order to properly and accurately validate and co-ordinate the sequence of activities performed on a system-wide basis. With reference to the association established between MCS functions and their required information, concurrent operation of functional activities is made possible in a flexible but controlled manner (via the Functional Interaction Manager). This enables interacting MCS software applications to run in parallel and their activities to be synchronised based on the availability of their required information which is monitored by appropriate mechanisms provided.

- ***Simplify and manage interconnection between interacting applications via use of a domain-specific IIS.***

  The MCS FIMM is built upon the CIM-BIOSYS IIS in order to utilise the more general integration services provided which includes inter-process communication and information management. This is to help simplify interconnection between applications by delivering these general services in a form which is more suited to typical users found in the MCS domains of manufacturing organisations. Furthermore, interaction knowledge is embodied in the MCS FIMM which is used at run-time to integrate MCS functions with human interaction enabled over the CIM-BIOSYS IIS, via the generic 'application shell'. Thus it is only necessary to link the applications concerned to this front-end common user interface in order to make use of the MCS FIMM services provided. This essentially removes the need to incorporate interaction knowledge in each individual application, thereby simplifying interconnection, facilitating interaction and providing opportunities to standardise and modularise MCS components.

Chapter 6

System Life Cycle Support

"The only Constant is change"
Bishop, PA Consulting Group (UK)

6.1 The Requirement for Integrated Life Cycle Support

In reality integrated manufacturing systems are characteristically evolutionary in nature and must be adaptable and responsive to changing needs. For example, a very common requirement is further integration with other functions and a re-engineering of existing functions in order to modify and enhance the functional capabilities provided. Change is essential in order to provide competitive differentiation and to ensure the survival of companies in the face of changing customer, supplier, financial and labour markets. As Tom Peters aptly pointed out in his highly publicised book entitled ***Thriving on Chaos,***

> *" To survive and become superlative in today's economic environment, the flexibility to react and be responsive to changes is highly desired....Impermanence is a cherished quality for excellent firms* " [Peters 1989]

Change in the requirements and characteristic properties of integrated manufacturing system will inevitably affect dependency relationships and information flows among interoperating MCS functions in the system. Hence the author strongly advocates the need to effectively support an integrated system through its life cycle, which will involve the following phases (refer to section 2.4 for details) :

- **Conceptual Design;** the prime focus is deciding what a system should do.
- **Detailed Design and Implementation;** this involves specifying how the global requirements defined can be realised in terms of building the required solutions.
- **Operation and Maintenance;** this characterises the working life of the installed solution, as well as necessary adjustments and repair during the operational lifetime of the system.

In order to facilitate ease of system development and change management, it is necessary to share and channel usable results and data between the different life cycle phases in a consistent and accurate manner [Singh and Weston 1994b]. However, *there is presently an absence of an integrated, formalised and structured approach which straddles the various life cycle phases, that can help support systems as they evolve* [Czernik and Quint 1992]. As is evident from the literature survey (refer to section 2.4.1), no one methodology includes a capability for modelling the functional, information, dynamic and decision aspects of integrated manufacturing control systems. As a result, *independent and separate use of a number of methods will be required if the formal modelling and construction of systems is required.*

As previously discussed (in section 2.4.2) the formal modelling of systems can provide an entry point for supporting the life cycle of manufacturing systems where *the models created (of function and information aspects) can serve as a source of knowledge during different life cycle phases*. However, it is necessary to develop additional life cycle support tools coupled closely to the modelling tool. Such a software toolset should exploit the knowledge contained within the model in order to :

- reference the functional and information models created during conceptual and detailed design;
- ensure compatibility and continuity between different life cycle phases, i.e. maintain consistency between models produced and used at each life cycle phase; and
- control and enforce structured implementation and change management.

Thus as part of this research study a new methodology and software toolset has been developed to support implementation, run-time and change processes, thereby facilitating integrated life cycle support of systems in response to changing needs. In this approach, system design and modelling methods, which typically provide a means of representing functional, information and behavioural views of a system, serve as the entry point. The information and function models created are exploited in downstream life cycle phases to ensure clarity, consistency, accuracy and re-utilisation of knowledge and data between phases.

6.2 Software Tools to Enact Function and Information Models

In order to enable functional and entity-attribute relationship modelling, IDEF modelling methodologies (namely $IDEF_0$ and $IDEF_{1X}$ respectively) were chosen as the starting point for this research because of the following reasons :

- They had demonstrated their usefulness as a 'simple and effective communication tool' which encourages end user involvement as well as their cooperation with system builders [Maji 1988].

- They have a growing popularity and acceptance, this being evident from the significant levels of research and industrial applications published in the literature. This reflects primarily their accessibility and potential in a wide range of applications.
- No other methodology known to the author claim to provide the same functional analysis capability [Colquhoun *et al.* 1993].
- These modelling methodologies offer opportunities for enhancement and integration with other tools [Colquhoun *et al.* 1993, Mayer and Painter 1991].
- Although the normal application of IDEF$_0$ and IDEF$_{1X}$ are separate and independent of each other, they are able to offer a formal and relatively complete representation of the manufacturing systems, this from their different modelling perspectives (i.e. functional and information).
- IDEF$_{1X}$ is a natural choice for representing a relational information model, as the model created has an inherent one-to-one correspondence with the entity-attribute relationships normally defined in a relational database [Arngrimsson and Vesterager 1992].

Figure 6-1 provides an overview of the MCS modelling and implementation environment conceived and used in this research to enact static MCS functional and information models. These models were created using IDEF$_0$ and IDEF$_{1X}$ methods. The purpose of this environment is to provide a formal and structured approach to

(i) facilitate implementation processes based on specific user requirements concerned with information systems comprising a number of relational data repositories;

(ii) enact the information and function models (which were created) in order to establish and configure associations, as identified and represented by the Function-Information Association Table of the MCS FIMM. This table being used to represent and encode system behaviour (see section 5.3 for details).

The environment itself is built from an aggregate of the following tools :

(A) EXPRESS to SQL Compiler
(B) STEP Parser
(C) IDEF$_{1X}$ to EXPRESS transformation tool
(D) IDEF$_{0/1X}$ Parser
(E) FIMM Configurator (refer to section 5.3 for details)

(A), (B) and (C) are tools which were developed by fellow researchers in the MSI Research Institute [Clements *et al.* 1993], whereas (D) and (E) were developed as part of this research study.

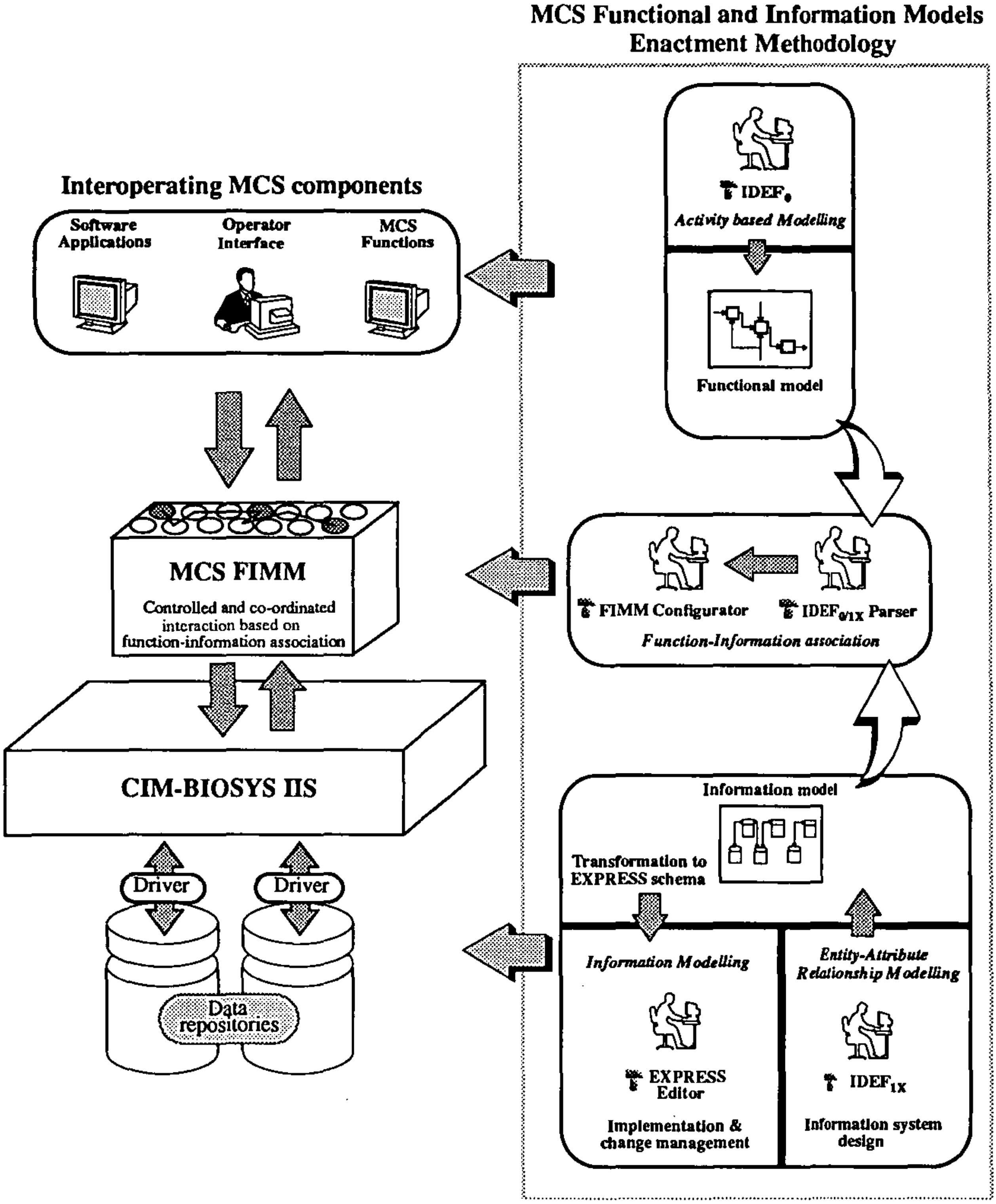

MCS Functional and Information Models Enactment Methodology
Interoperating MCS components
Software Applications
Operator Interface
MCS Functions
IDEF$_0$
Activity based Modelling
Functional model
MCS FIMM
Controlled and co-ordinated interaction based on function-information association
FIMM Configurator
IDEF$_{0/1X}$ Parser
Function-Information association
CIM-BIOSYS IIS
Information model
Transformation to EXPRESS schema
Driver
Driver
Data repositories
Information Modelling
Entity-Attribute Relationship Modelling
EXPRESS Editor
IDEF$_{1X}$
Implementation & change management
Information system design

Figure 6-1 : Software toolset for integrated life cycle support

]

6.2.1 Information Model Enactment

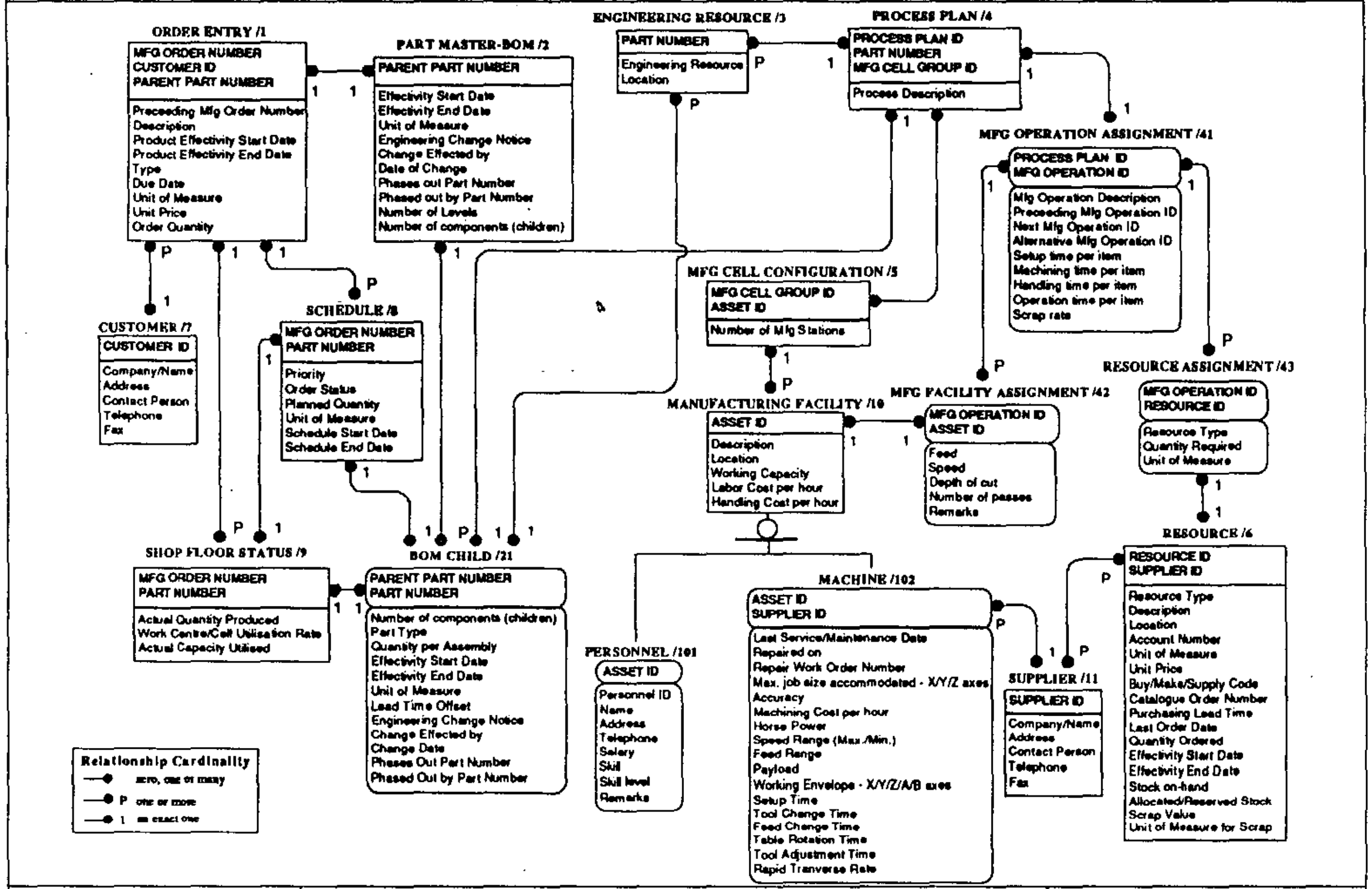
The IDEF_{1X} entity-attribute relationship modelling tool [ICAM 1985] was used to represent the global schema of common information used by MCS. It is a composite view of a set of information models which corresponds to the generic reference models (as identified and described by the author in section 4.2). The structure, content and entity relationships among the attributes of the information models are clearly described. The semantics are, therefore, made explicit so that there is common interpretation of the relationships among data items. Please refer to Appendix XII and Figure 6-2 for an overview of IDEF_{1X} and the entity-attribute relationships of the information models respectively.

The EXPRESS data modelling language [Schenck 1989] is an emerging standard which is used within ongoing STEP (STandard for the Exchange of Product data) [ISO 1993, Joris *et al.* 1993] initiatives world-wide to define data models. EXPRESS is used in this research to facilitate detailed information modelling and was chosen because the author believed that it would enable implementation of a physical system in a manner which maintains the structure of the formal models. Through parallel research effort at MSI the following tools have been developed to exploit EXPRESS as a means of representing aspects of the information architecture of CIM systems and of providing a means of managing change more effectively [Clements *et al.* 1993, Clements 1992, Clements *et al.* 1991, Clements 1991a, Clements 1991b] :

- **EXPRESS to SQL Compiler**
  Using an EXPRESS to SQL Compiler software tool created at MSI by Clements, an EXPRESS information model can be directly compiled to generate SQL statements which will later be responsible for creating SQL compliant tables within the database. In this way the database is structured according to EXPRESS defined schemas so that relationships are strictly maintained; relevant information concerning the tables and their interrelationships are automatically generated and stored in a dictionary. This approach is database independent (EXPRESS is not biased towards implementation) and can produce as output SQL statements which function for different relational database implementations, such as Ingres and ORACLE.

- **STEP Parser**
  This tool has been created by Clements to enable the population of the tables (within different relational databases) with real data in the format specified by the EXPRESS model. This process is structured by information contained in the output files generated by the EXPRESS to SQL complier.

USED AT:	AUTHOR: VALDEW SINGH PROJECT: Definition of Information models entity-attribute relationships NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 15/05/93 REV:		WORKING	READER DATE	CONTEXT:
				DRAFT		
			X	RECOMMENDED		
				PUBLICATION		



NODE: IDEF1X	TITLE: Information Models Entity-Attribute relationships	NUMBER:

Figure 6-2 : IDEF$_{1X}$ entity-attribute relationship model representing a composite view of the information models

In addition an IDEF$_{1X}$ **to EXPRESS transformation tool** was developed by the author to **enable the automatic creation of an EXPRESS based information model from IDEF$_{1X}$ entity-attribute relationship descriptions.** It operates by directly mapping the entities of the IDEF$_{1X}$ model into entities of the EXPRESS model. The reader should refer to Appendix XIV for details on the EXPRESS based information model in which the global schema is described. The establishment of this IDEF$_{1X}$ to EXPRESS computerised link not only offers a means of formally structuring information requirements but also supports design, build and change processes associated with them.

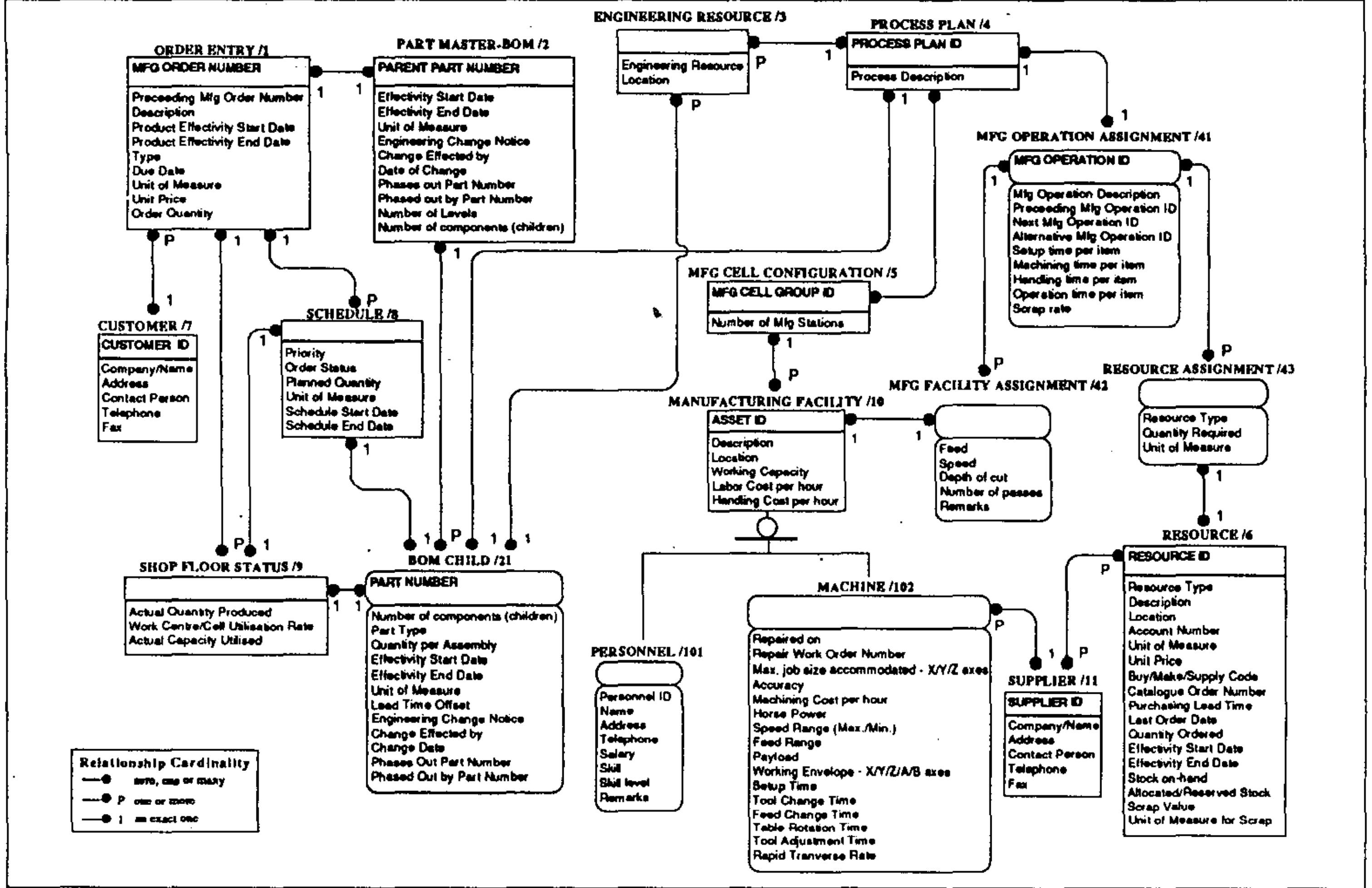
The overall approach conceived by the author offers the following advantages :

- *It helps simplify entity-attribute relationship modelling.*
  This is possible because EXPRESS has a strong object oriented nature where the information models are treated as objects, thus allowing modularity and data inheritance. It reduces the tedium and complexity associated with the schematisation of relational information models by eliminating the need to specify all common or primary attributes (which are necessary to link the tables together). Instead relationships among tables can be readily (i) established by allowing tables to inherit the attributes of other tables to which they are linked (as illustrated in Figure 6-3), and (ii) altered and relationships added or removed without causing chaos in what remains.
- *Allows easy extension and modification of information models.*
- *Supports and encourages the reusability of information objects, thereby reducing the development time and enabling wider scope solutions.*
- *Enables the use of IDEF$_{1X}$ created models, this being important in system development and maintenance phases of projects.*
- *Implementation is database independent.*

6.2.2 MCS Functional Modelling

Within the context of this research study, the process of functional modelling is concerned with much broader issues related to formal identification and description of dependencies among functional activities, rather than focusing on specific details pertaining to the capabilities and requirements of each functional activity.

USED AT:	AUTHOR: VALDEW SINGH	DATE: 24/05/93		WORKING	READER DATE	CONTEXT:
	PROJECT: Information models entity-attribute relationships for input to EXPRESS	REV:		DRAFT		
			X	RECOMMENDED		
	NOTES: 1 2 3 4 5 6 7 8 9 10			PUBLICATION		



NODE: IDEF1X	TITLE: **Information Models Entity-Attribute relationships for Input to EXPRESS**	NUMBER:

Figure 6-3 : IDEF$_{1X}$ entity-attribute relationship model modified for translation to EXPRESS schema

A proprietary IDEF$_0$ activity based modelling tool [Meta Software 1990] was used in this study to produce a function model from which, in a structured manner, identifies and defines dependencies and inter-relationships among Production planning, Finite capacity scheduling, Process planning and Cell control activities. A further decomposition of this functional model (in a hierarchical manner) can be realised to define more closely the interactions among the functional activities where inputs required to drive the functions and their outputs (generated under supervision of their controls and enabling mechanisms) are represented. The hierarchical decomposition of this functional model is illustrated in Figures 6-4, 6-5 and 6-6. Please refer to Appendix XII for an overview of IDEF$_0$.

6.2.3 System Behaviour Enactment

A formal definition of the interaction processes between the functional components of an integrated system requires complete and accurate descriptions of (i) the flow of information between function blocks; and (ii) the form and type of information; which need to be made available to support and drive those functions, so that they can realise their assigned tasks. As previously discussed (in Section 2.5.2) there should be a means of

(a) unifying the perspectives of functional and information modelling in order to establish an association between the two modelling streams; and

(b) facilitating the structuring of downstream life cycle processes, for example to aid implementation and configuration in relation to co-ordination and control for functional interaction.

Hence a novel approach is offered in this research study to meet these requirements. This approach involves use of an IDEF$_{0/1X}$ Parser tool and the FIMM Configurator (previously described) which together can formally define and describe the behavioural aspects of an interoperating system where the

- IDEF$_{0/1X}$ Parser is responsible for the enactment of MCS function and information models in order to establish association between them.
- FIMM Configurator generates data which encodes the required associations (refer to section 5.3 for further details on FIMM Configurator).

As the IDEF$_{0/1X}$ Parser and FIMM Configurator tools need to operate in close relation with each other, they share the same database (i.e. FIMM database). Within this database all relevant data concerning the functions, information entities and their nature of association (i.e. as either input or output) are stored.

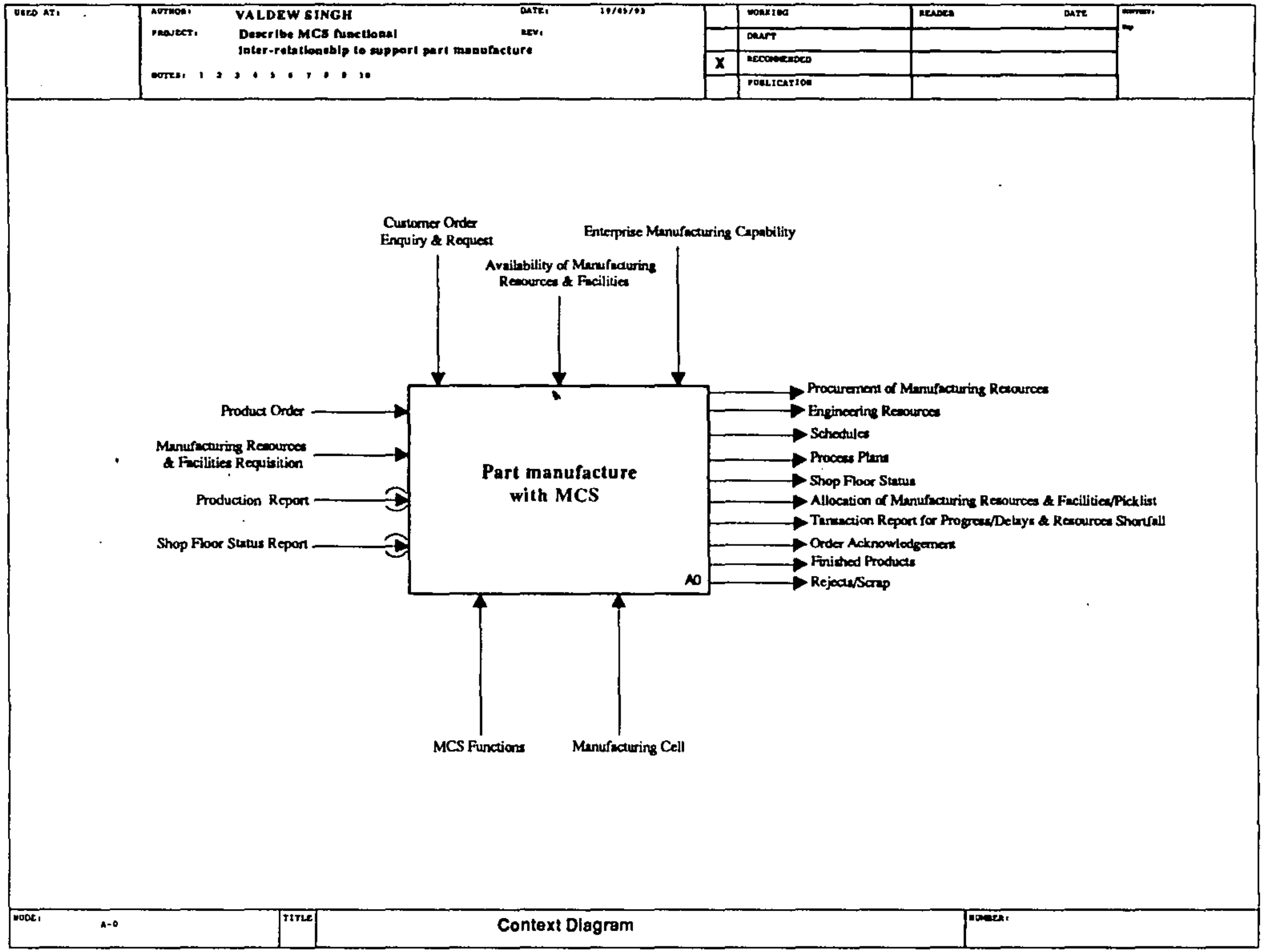

USED AT:
AUTHOR: VALDEW SINGH
DATE: 19/05/93
PROJECT: Describe MCS functional inter-relationship to support part manufacture
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
X RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
Top
Customer Order Enquiry & Request
Availability of Manufacturing Resources & Facilities
Enterprise Manufacturing Capability
Product Order
Manufacturing Resources & Facilities Requisition
Production Report
Shop Floor Status Report
Part manufacture with MCS
A0
Procurement of Manufacturing Resources
Engineering Resources
Schedules
Process Plans
Shop Floor Status
Allocation of Manufacturing Resources & Facilities/Picklist
Tansaction Report for Progress/Delays & Resources Shortfall
Order Acknowledgement
Finished Products
Rejects/Scrap
MCS Functions
Manufacturing Cell
NODE: A-0
TITLE: Context Diagram
NUMBER:

Figure 6-4 : IDEF$_0$ function model (Context Diagram)

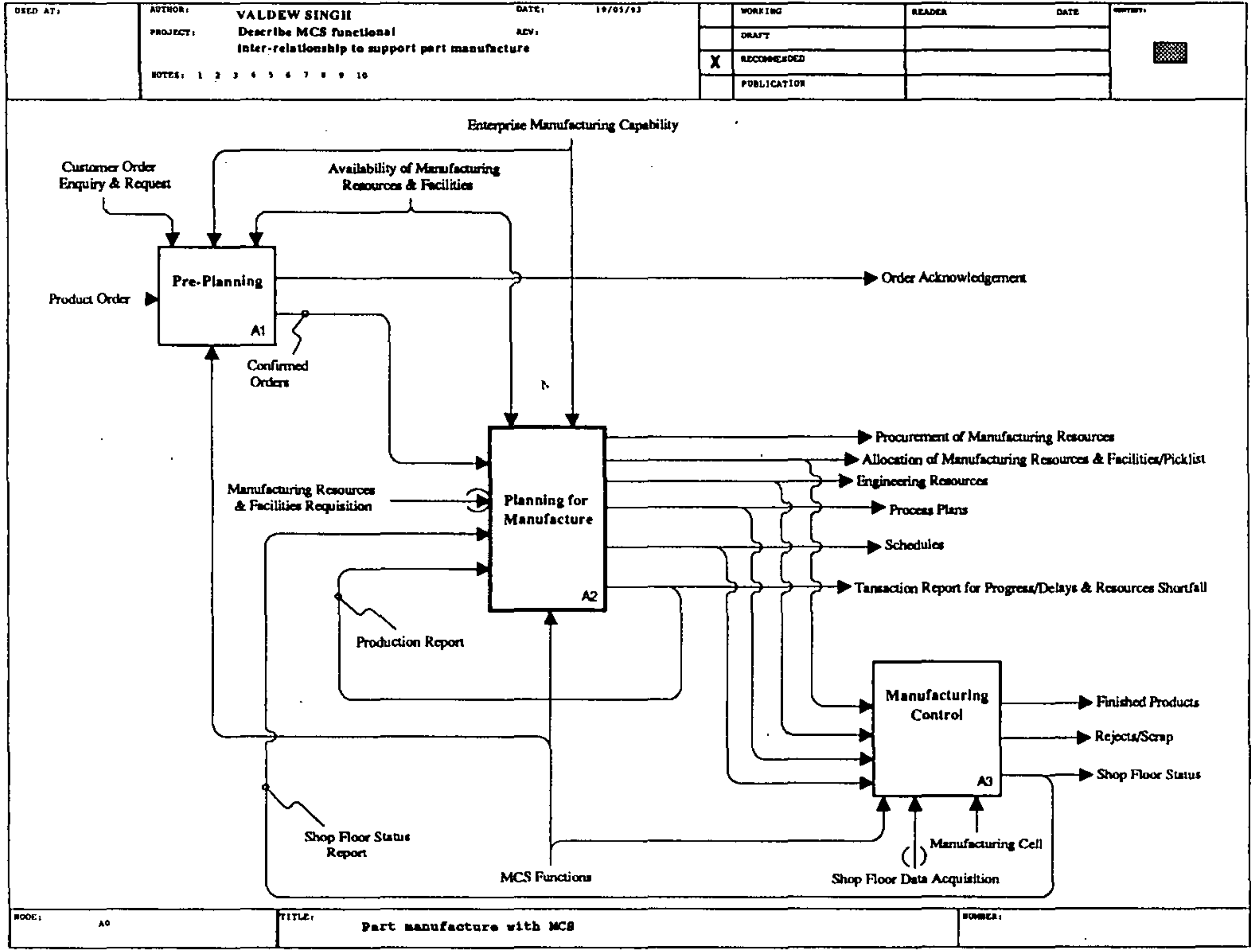
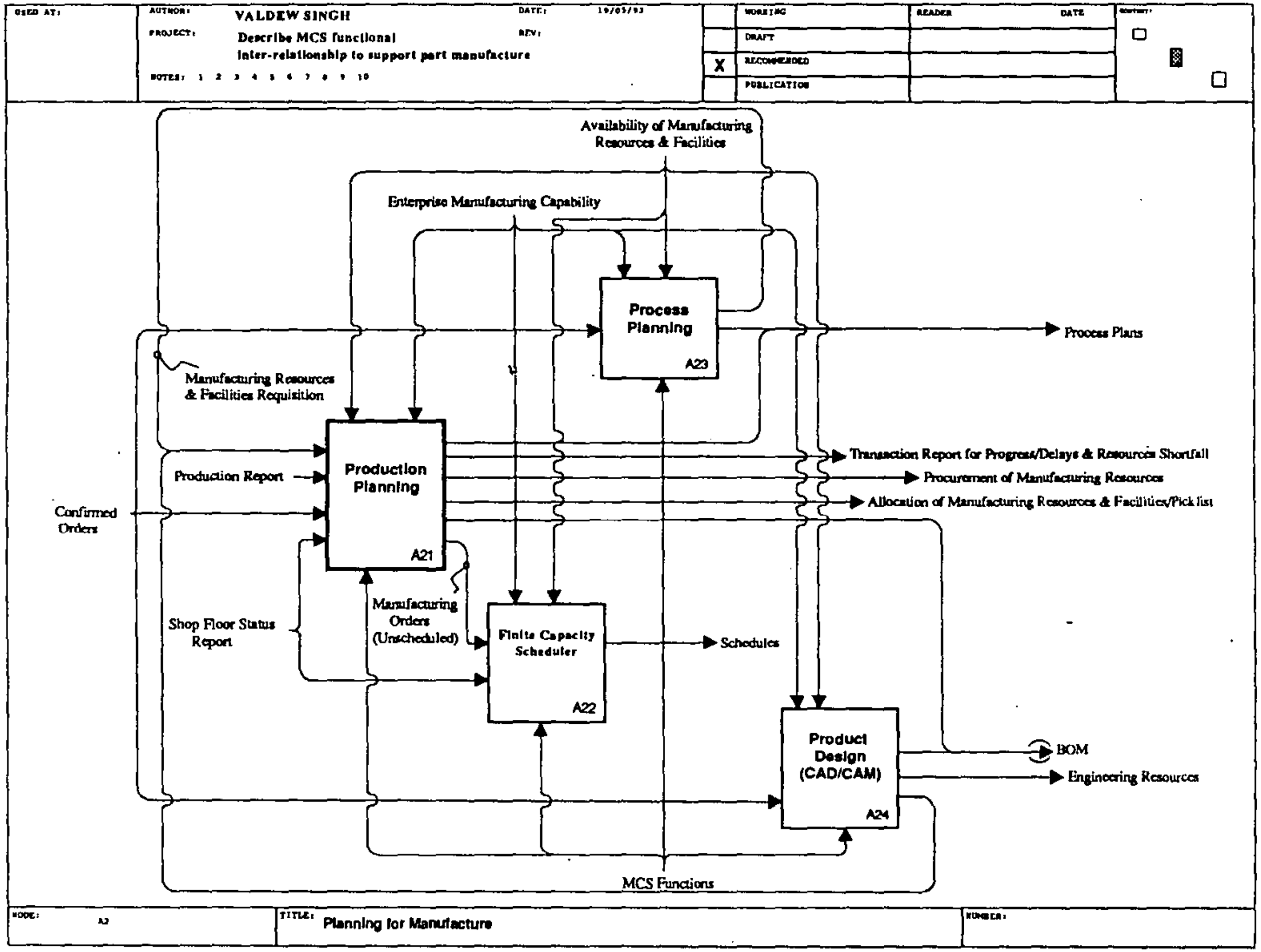

USED AT:
AUTHOR: VALDEW SINGH
DATE: 19/03/93
PROJECT: Describe MCS functional inter-relationship to support part manufacture
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
X RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
Enterprise Manufacturing Capability
Customer Order Enquiry & Request
Availability of Manufacturing Resources & Facilities
Product Order
Pre-Planning
A1
Order Acknowledgement
Confirmed Orders
Manufacturing Resources & Facilities Requisition
Planning for Manufacture
A2
Procurement of Manufacturing Resources
Allocation of Manufacturing Resources & Facilities/Picklist
Engineering Resources
Process Plans
Schedules
Tansaction Report for Progress/Delays & Resources Shortfall
Production Report
Manufacturing Control
A3
Finished Products
Rejects/Scrap
Shop Floor Status
Shop Floor Status Report
MCS Functions
Manufacturing Cell
Shop Floor Data Acquisition
NODE: A0
TITLE: Part manufacture with MCS
NUMBER:

Figure 6-5 : IDEF$_0$ function model to describe Production Planning and its inter-relationship with other MCS functions



USED AT:

AUTHOR: VALDEW SINGH DATE: 19/05/93

PROJECT: Describe MCS functional inter-relationship to support part manufacture REV:

NOTES: 1 2 3 4 5 6 7 8 9 10

	WORKING
	DRAFT
X	RECOMMENDED
	PUBLICATION

READER DATE

CONTEXT: □ □

NODE: A2

TITLE: Planning for Manufacture

NUMBER:

Availability of Manufacturing Resources & Facilities
Enterprise Manufacturing Capability
Process Planning
A23
Process Plans
Manufacturing Resources & Facilities Requisition
Production Planning
A21
Production Report
Confirmed Orders
Transaction Report for Progress/Delays & Resources Shortfall
Procurement of Manufacturing Resources
Allocation of Manufacturing Resources & Facilities/Pick list
Shop Floor Status Report
Manufacturing Orders (Unscheduled)
Finite Capacity Scheduler
A22
Schedules
Product Design (CAD/CAM)
A24
BOM
Engineering Resources
MCS Functions

Figure 6-6 : IDEF$_0$ function model with detailed level of decompositon for Planning for Manufacture activity within Production Planning domain

As illustrated in Figure 6-7, the tools perform the following specific tasks :

- **$\text{IDEF}_{0/1X}$ Parser**

  This sorts through and assimilates large amounts of data generated in the IDEF_0 and IDEF_{1X} reports which contain details pertaining to the models. The following services are provided :

  - selection of all functions defined in the IDEF_0 functional model;
  - selection of entity-attributes, which comprise the various information models defined with the IDEF_{1X} tool;
  - interactive selection of required functions and information entities through specially developed user-friendly interfaces (as illustrated in Appendix XVII), where the selected functions are assigned unique identifiers for easy representation; and
  - populates selected data into the FIMM database.

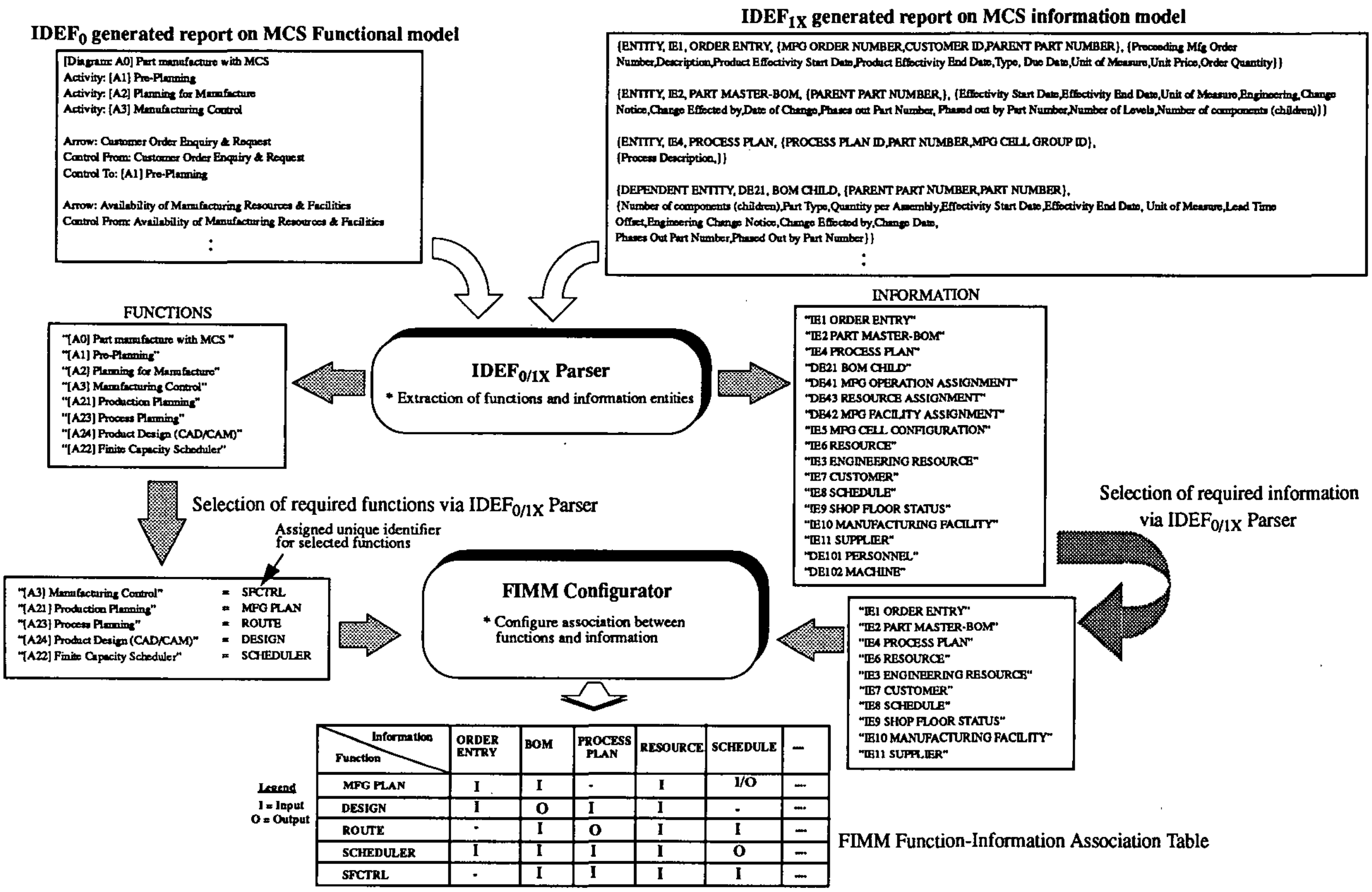
  Program listings for the IDEF_{1X} Parser implemented by the author are included in Appendix XVIII.

- **FIMM Configurator**

  The FIMM Configurator processes the data derived from the $\text{IDEF}_{0/1X}$ Parser. It offers services which enable the user to interactively set, edit and display associations between functions and information. The configured associations between functions and information entities are stored in the Function-Information Association Table of the MCS FIMM and used (during run-time) for the following purposes (see section 5.3) :

  - to ensure accountability of information to functions; and
  - to control the flow of sequence among functions in a controlled and co-ordinated manner where information availability and completeness is verified prior to enabling any function requiring access to it.

The $\text{IDEF}_{0/1X}$ Parser and FIMM Configurator have been applied to the IDEF_0 function model (illustrated in Figures 6-4, 6-5 and 6-6) and the IDEF_{1X} entity-attribute relationship model (illustrated in Figure 6-2); this by processing their generated reports (included in Appendices XVI and XV respectively). This proof-of-concept demonstration illustrated the ability to enable and support the establishment, management and change of associations between functions and information as indicated in Figure 6-7.



IDEF$_0$ generated report on MCS Functional model

[Diagram: A0] Part manufacture with MCS
Activity: [A1] Pre-Planning
Activity: [A2] Planning for Manufacture
Activity: [A3] Manufacturing Control

Arrow: Customer Order Enquiry & Request
Control From: Customer Order Enquiry & Request
Control To: [A1] Pre-Planning

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
:

IDEF$_{1X}$ generated report on MCS information model

{ENTITY, IE1, ORDER ENTRY, {MFG ORDER NUMBER,CUSTOMER ID,PARENT PART NUMBER}, {Preceeding Mfg Order Number,Description,Product Effectivity Start Date,Product Effectivity End Date,Type, Due Date,Unit of Measure,Unit Price,Order Quantity}}

{ENTITY, IE2, PART MASTER-BOM, {PARENT PART NUMBER,}, {Effectivity Start Date,Effectivity End Date,Unit of Measure,Engineering,Change Notice,Change Effected by,Date of Change,Phases out Part Number, Phased out by Part Number,Number of Levels,Number of components (children)}}

{ENTITY, IE4, PROCESS PLAN, {PROCESS PLAN ID,PART NUMBER,MFG CELL GROUP ID},
{Process Description,}}

{DEPENDENT ENTITY, DE21, BOM CHILD, {PARENT PART NUMBER,PART NUMBER},
{Number of components (children),Part Type,Quantity per Assembly,Effectivity Start Date,Effectivity End Date, Unit of Measure,Lead Time Offset,Engineering Change Notice,Change Effected by,Change Date,
Phases Out Part Number,Phased Out by Part Number}}
:

FUNCTIONS

"[A0] Part manufacture with MCS "
"[A1] Pre-Planning"
"[A2] Planning for Manufacture"
"[A3] Manufacturing Control"
"[A21] Production Planning"
"[A23] Process Planning"
"[A24] Product Design (CAD/CAM)"
"[A22] Finite Capacity Scheduler"

IDEF$_{0/1X}$ Parser
* Extraction of functions and information entities

INFORMATION

"IE1 ORDER ENTRY"
"IE2 PART MASTER-BOM"
"IE4 PROCESS PLAN"
"DE21 BOM CHILD"
"DE41 MFG OPERATION ASSIGNMENT"
"DE43 RESOURCE ASSIGNMENT"
"DE42 MFG FACILITY ASSIGNMENT"
"IE5 MFG CELL CONFIGURATION"
"IE6 RESOURCE"
"IE3 ENGINEERING RESOURCE"
"IE7 CUSTOMER"
"IE8 SCHEDULE"
"IE9 SHOP FLOOR STATUS"
"IE10 MANUFACTURING FACILITY"
"IE11 SUPPLIER"
"DE101 PERSONNEL"
"DE102 MACHINE"

Selection of required functions via IDEF$_{0/1X}$ Parser

Assigned unique identifier for selected functions

Selection of required information via IDEF$_{0/1X}$ Parser

"[A3] Manufacturing Control"	=	SFCTRL
"[A21] Production Planning"	=	MFG PLAN
"[A23] Process Planning"	=	ROUTE
"[A24] Product Design (CAD/CAM)"	=	DESIGN
"[A22] Finite Capacity Scheduler"	=	SCHEDULER

FIMM Configurator
* Configure association between functions and information

"IE1 ORDER ENTRY"
"IE2 PART MASTER-BOM"
"IE4 PROCESS PLAN"
"IE6 RESOURCE"
"IE3 ENGINEERING RESOURCE"
"IE7 CUSTOMER"
"IE8 SCHEDULE"
"IE9 SHOP FLOOR STATUS"
"IE10 MANUFACTURING FACILITY"
"IE11 SUPPLIER"

Legend
I = Input
O = Output

Information / Function	ORDER ENTRY	BOM	PROCESS PLAN	RESOURCE	SCHEDULE	---
MFG PLAN	I	I	-	I	I/O	---
DESIGN	I	O	I	I	-	---
ROUTE	-	I	O	I	I	---
SCHEDULER	I	I	I	I	O	---
SFCTRL	-	I	I	I	I	---

FIMM Function-Information Association Table

Figure 6-7 : Methodology for function-information association

The methodology proposed in this research study is based on the use of $\mathrm{IDEF}_{0/1X}$ Parser and FIMM Configurator tools and offers an effective and formal means of capturing and establishing a link between MCS functions and their required information. *No extra effort is placed on the system designer or builder, except for configuring the required association*s based on data drawn directly from existing function and information models. In addition, the consistency and accuracy of the associations between function and information is ensured and maintained from a very early stage in the system life cycle (i.e. the conceptual design stage) right through to system implementation and maintenance. This reduces uncertainty and resolves many potential conflicts. In addition, this approach also helps improve the accountability and traceability of functions and information.

6.3 Summary

As illustrated by this research study, system design and modelling can provide an effective entry point for supporting the life cycle of systems in an integrated manner (i.e. in a manner in which the results and knowledge generated at one phase can be used during other life cycle phases). Models generated using one or more chosen modelling methods can provide a formal representation of different views (i.e. function, information, behaviour) of the system under consideration and can serve as a source of usable data to structure and enable various downstream life cycle processes.

However, in order to truly achieve and enable integrated life cycle support it is necessary to allow usable data to be referenced, accessed, manipulated and formatted in a manner suitable for use at each life cycle phase. The work reported in this chapter represents a useful first step towards meeting this requirement, namely through use of the following software toolset:

- **EXPRESS to SQL Compiler**
- **STEP Parser**
- **IDEF_{1X} to EXPRESS transformation tool**
- **$\mathrm{IDEF}_{0/1X}$ Parser**
- **FIMM Configurator**

Each of these tools exists as an independent entity but are linked through a shared database (i.e. FIMM database) where usable data is stored for common access and usage to support the different life cycle phases. It should be noted that the software toolset developed can be configured to support other system design and modelling methodologies on condition that the methodologies in question provide ready access to their underlying data and knowledge structures which they use to encapsulate function and information models.

Finally, in relation to the overall methodology conceived in this research study to enable interoperation of MCS components (see section 3.5), this chapter has reported on two sub-methods, namely :

- **Means of Enacting Function Models**
- **Means of Enacting Information Models**

In view of their need to adapt and respond to changes in system requirements, the set of build tools used here offers the system builder a more formalised and structured approach (as compared to current practice) to the creation and maintenance of integrated manufacturing systems.

Chapter 7

Use and Appraisal of the Methodology Derived

7.1 Proof-of-Concept MCS Implementation

A proof-of-concept MCS implementation study was carried out in order to illustrate the application and ascertain the level of effectiveness of the methodology derived in this research study, which seeks to enable interoperability among MCS components.

In this study a proof-of-concept MCS system was built to demonstrate the interoperation of a number of typical software applications which are representative of the manufacturing control domain being considered in this research study [Singh and Weston 1994a, 1994b, 1993]. The main generic class of problem tackled in the system involves the transformation of a commercially available stand-alone proprietary CAPM package, namely MCC, into a more open system so as to enable interoperation with other MCS components. This demonstrates a particularly important industrial capability, namely the integration of existing legacy (or "as is") software applications. The following sub-systems are utilised :

- MCS FIMM (Functional Interaction Management Module) to facilitate MCS domain specific functional interaction and to govern system behaviour (during run-time).
- CIM-BIOSYS IIS to simplify interconnection and to provide common general purpose integration services to support functional interaction.
- Set of build tools (used and developed in this research study) to formally structure
  - interaction among MCS components;
  - implement and maintain information system on a system-wide basis, by managing and maintaining a data repository which holds information of common concern.

7.2 MCS Software Interoperability Demonstration System

As illustrated in Figure 7-1, the following functional components are involved in the MCS demonstration system :

- **Production Planning**

  This encompasses order entry, scheduling, and manufacturing resource and facility management which includes allocation, procurement as well as routing for part manufacture. MCC (Manufacturing Control Code), which is a commercially available CAPM package from John Brown Systems PLC, was chosen for this study as its information is stored in a relational database, namely ORACLE RDBMS, and is to a certain degree accessible by other applications. The purpose of MCC is to turn product orders into manufacturing schedules for enaction by a set of manufacturing resources. Please refer to Appendix XIX for an overview of MCC.

- **Finite Capacity Scheduler**

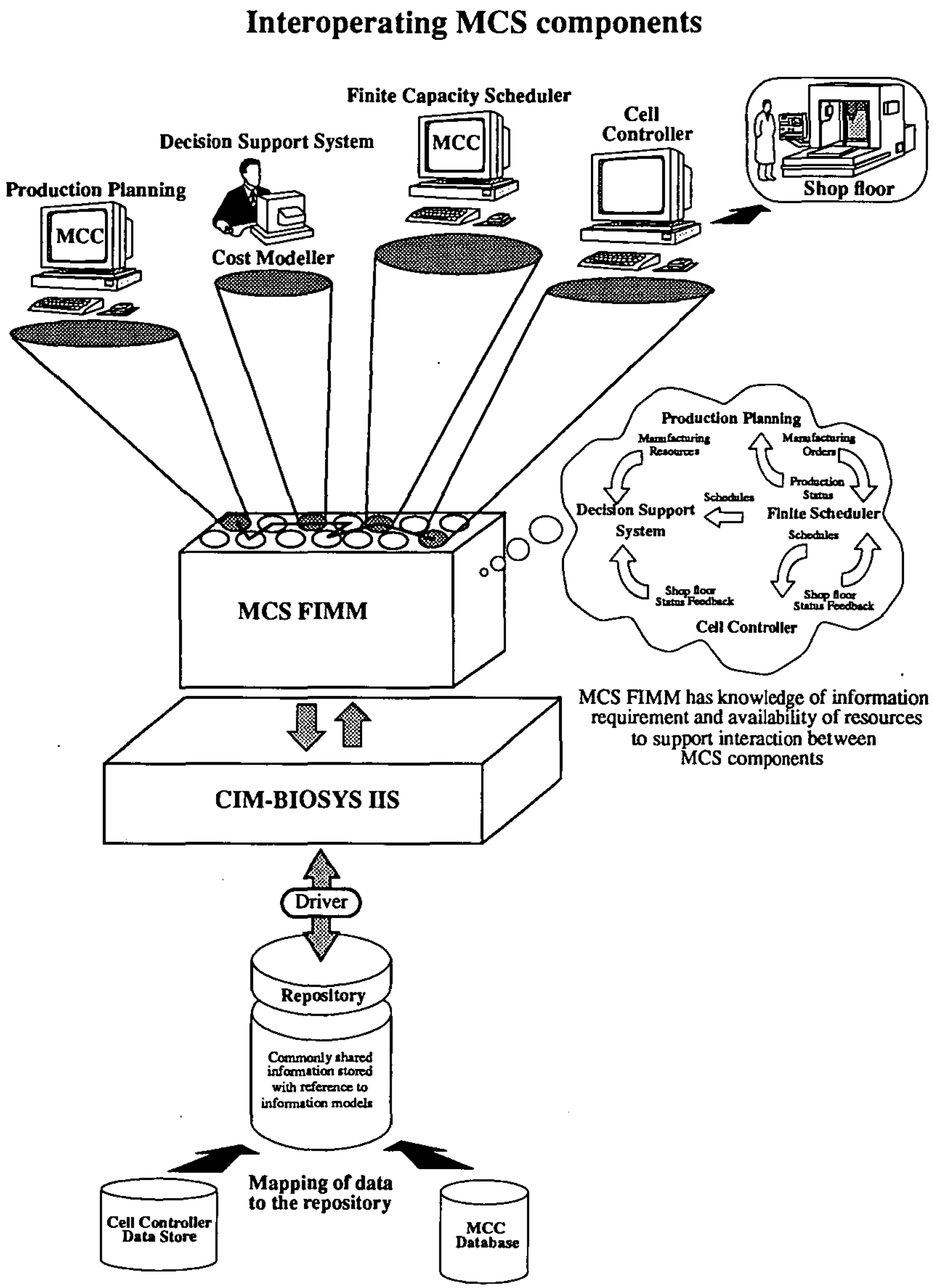
  Although the finite capacity scheduling function available in MCC is utilised, it is intentionally treated as an independent and separate function in order to isolate and effectively study its interaction needs with other MCS components. This finite capacity scheduling sub-system is responsible for the short term planning of manufacturing orders in a manner which optimises manufacturing operations on the shop floor. It relies on the availability of data from production planning, treating it as its input, to perform the necessary scheduling function based on, for example: resource and manufacturing facility allocation and constraints, part procurement and manufacture lead time as well as routes for part manufacture.

- **Cell Controller**

  A cell controller, separately conceived and developed by a co-MSI researcher, Binglu Zhang, was integrated into the MCS demonstration system. Here the cell controller has responsibility for a segment of a pseudo shop floor (based on data supplied by a local engineering company manufacturing cranes and hoists) and is required to despatch planned manufacturing orders (scheduled by the finite capacity scheduler), and to co-ordinate, execute, control and monitor the operation of shop floor activities. The cell controller is also responsible for shop floor data acquisition thereby enabling and generating production status feedback.

- **Decision Support System**

  This class of software system was represented by a dynamic cost analysis tool which was created by another co-MSI researcher [Shaharoun 1992] to support strategic decision making for economic part manufacture. This decision support system has been included in the demonstration system in order to highlight the ease with which the system can be



MCS FIMM has knowledge of information requirement and availability of resources to support interaction between MCS components

Figure 7-1 : Demonstration system for MCS software interoperability

expanded in terms of building and plugging in ad-hoc specialised MCS components that require access to data available in the system data repository. It relies on both planned and actual production data, derived from production planning and the cell controller respectively, to perform the necessary cost analysis.

It should be noted that the cell controller and the decision support system for cost analysis represent a new generation of modular and reconfigurable MCS components created through research effort at Loughborough's MSI Research Institute.

The MCS demonstration system was built to facilitate interoperation between the above-mentioned functional components in a manner to enable sharing of and access to information of common interest (see Figure 7-2 which depicts information flow between functional components and dependency between information entities).

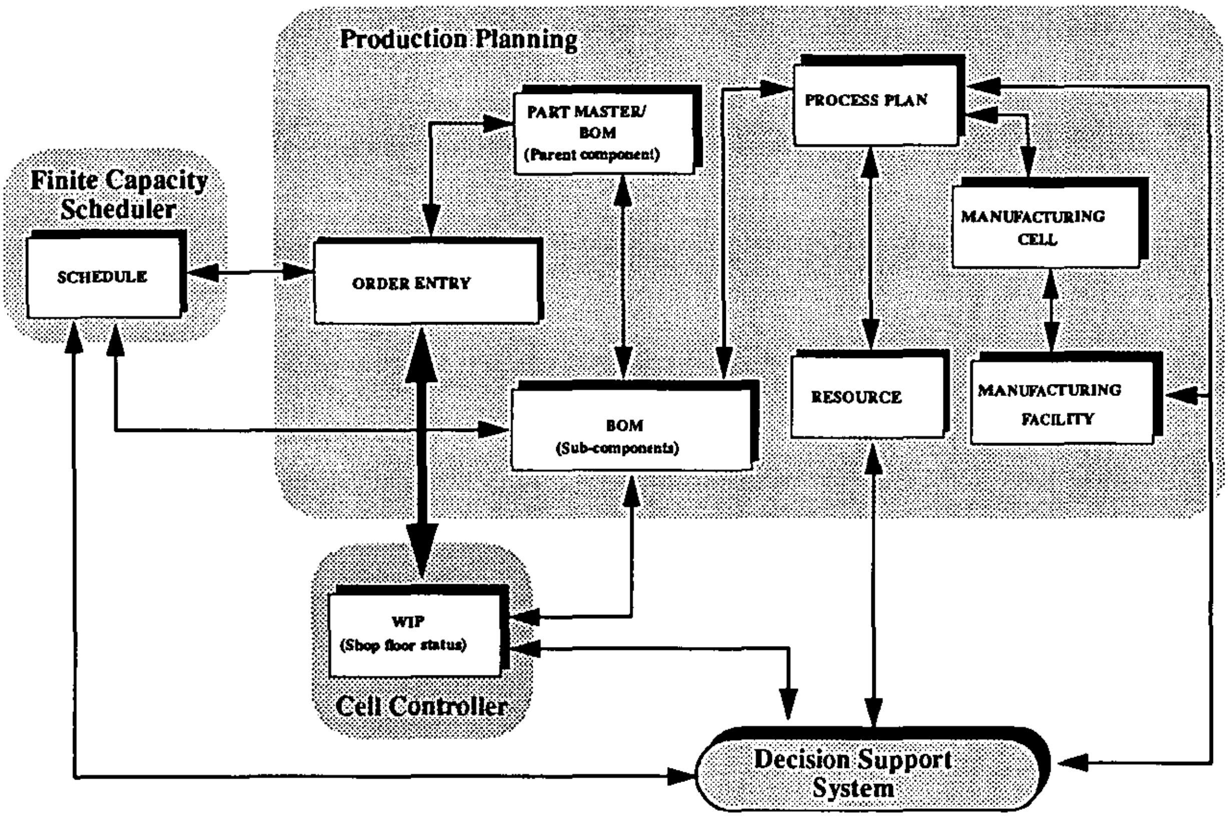


Figure 7-2 : Overview of information flow and dependency between information entities

7.2.1 Implementation of the Demonstration System

The following four inter-related meta-steps were supported by the author's methodology in a way which structured the development of the proof-of-concept MCS demonstration system :

- Implementation of a system-wide information repository
- Flexible interconnection of the functional components of the MCS
- Establishment and configuration of appropriate functional interaction capability
- Establishment and configuration of user interface capabilities

Details of the activities carried out in each of these meta-steps are outlined as follows :

(I) Implementation of a system-wide information repository

Here the following activities are facilitated which are representative example activities typically involved in establishing and managing the manufacturing information system :

(a) Identification and formal representation of information requirements

A global schema corresponding to system-wide shared information, i.e. shared between the MCS components concerned, was represented in the proof-of-concept system with reference to the generic information models. It represents a unification of information entities of common concern (to be held in the system data repository) which will require to be accessed and updated. Here IDEF_{1X} was used to formally represent the global schema (as illustrated in Figure 7-3). It uncovers semantic properties of the underlying information system and make them explicit within the data definition of information objects.

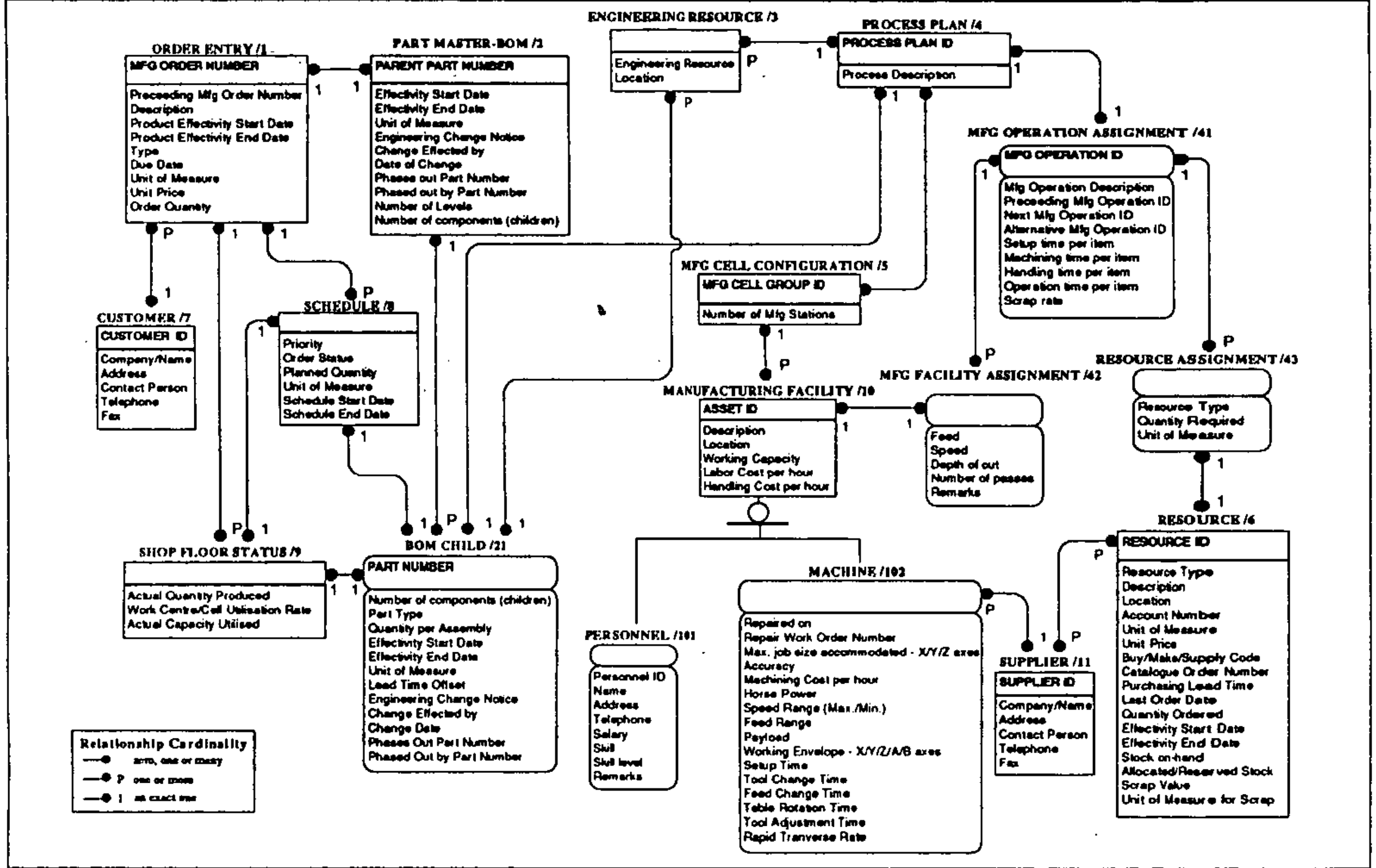
(b) Mapping between local databases to the system data repository

Proprietary information, stored in the MCC relational database, which is of common interest to other MCS functional components, were mapped onto the system data repository via reference to the global schema (refer to Figures 7-4, 7-5 and 7-6 for illustration). This same mapping principle can be applied for all legacy components, in which proprietary information data structure is used, thereby enabling other system components to gain access and update that information in a flexible manner.

(c) Creation of relational tables stored in the system data repository

The set of build tools used here to develop the system data repository is depicted in Figure 7-7. First the **IDEF_{1X} to EXPRESS transformation tool** is applied to automatically create an EXPRESS based information model which has an underlying data structure related to the entity relationships defined in the global schema, i.e.

USED AT:	AUTHOR: VALDEW SINGH PROJECT: Information models entity-attribute relationships for input to EXPRESS NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 24/05/93 REV:	WORKING DRAFT ☒ RECOMMENDED PUBLICATION	READER DATE	CONTEXT:



NODE: IDEF1X	TITLE: **Information Models Entity-Attribute relationships for Input to EXPRESS**	NUMBER:

Figure 7-3 : IDEF$_{1X}$ entity-attribute relationship model representing the global schema

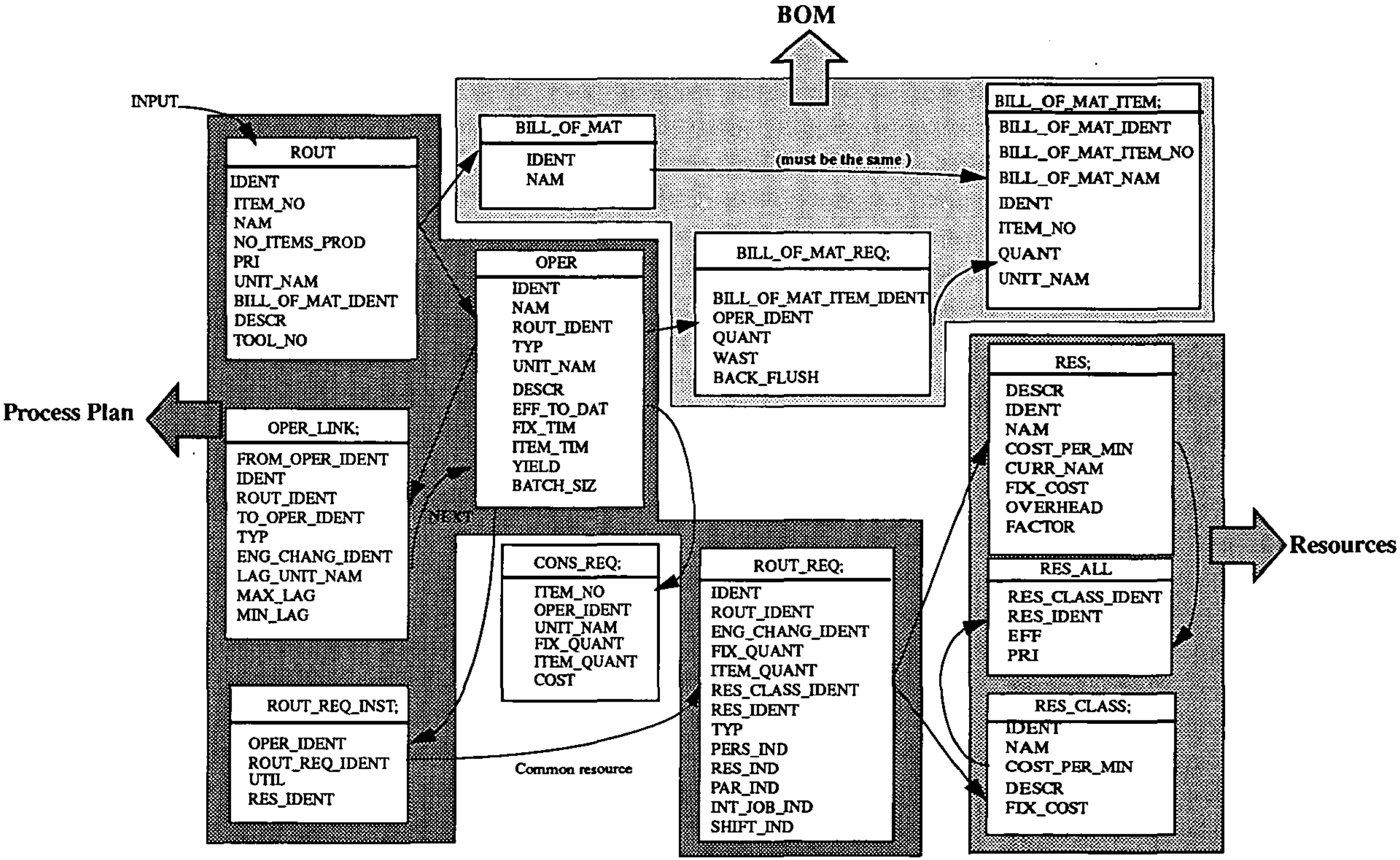


Figure 7-4 : Grouping of product and process information represented in MCC database with reference to information models

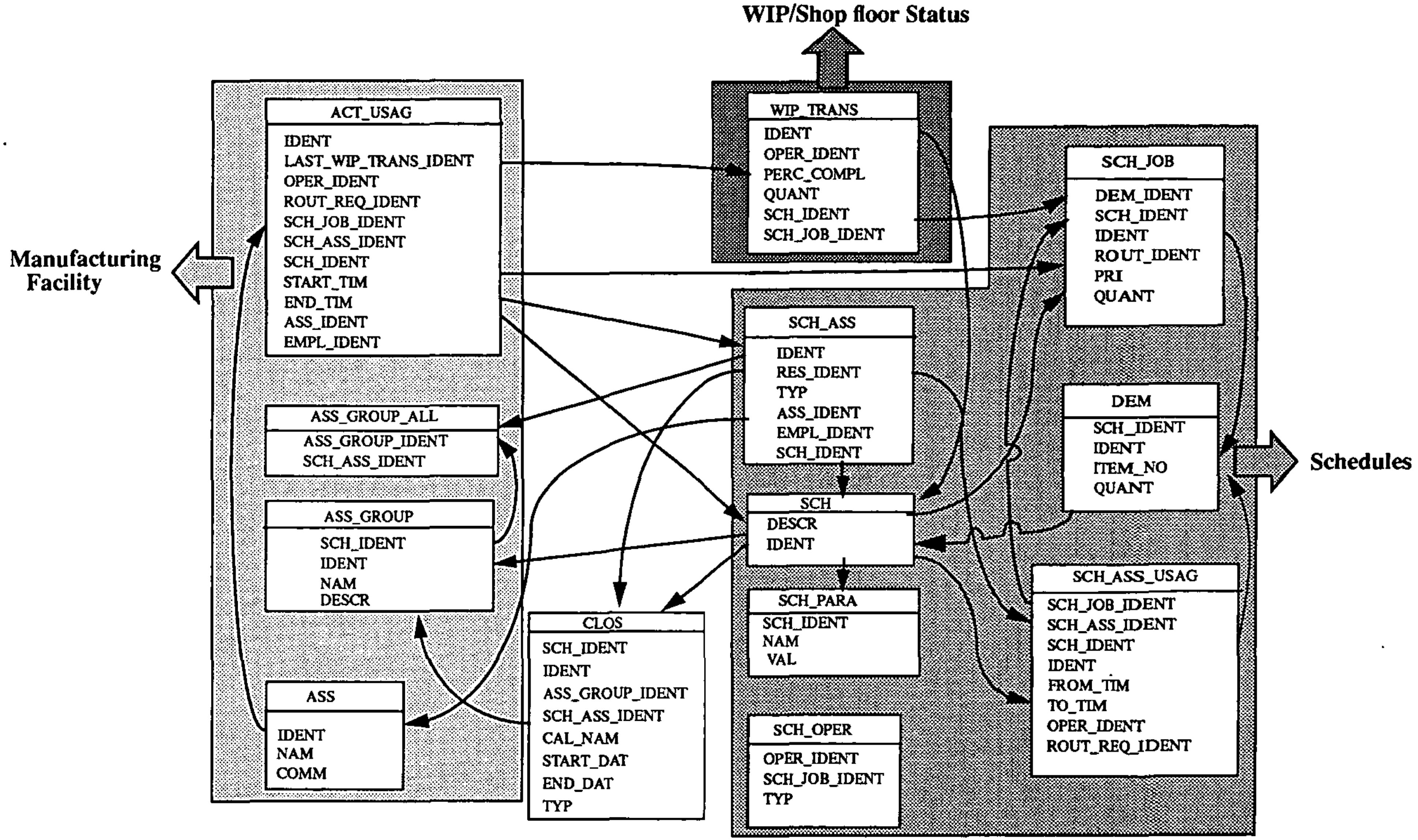


Figure 7-5 : Grouping of schedule information represented in MCC database with reference to information models

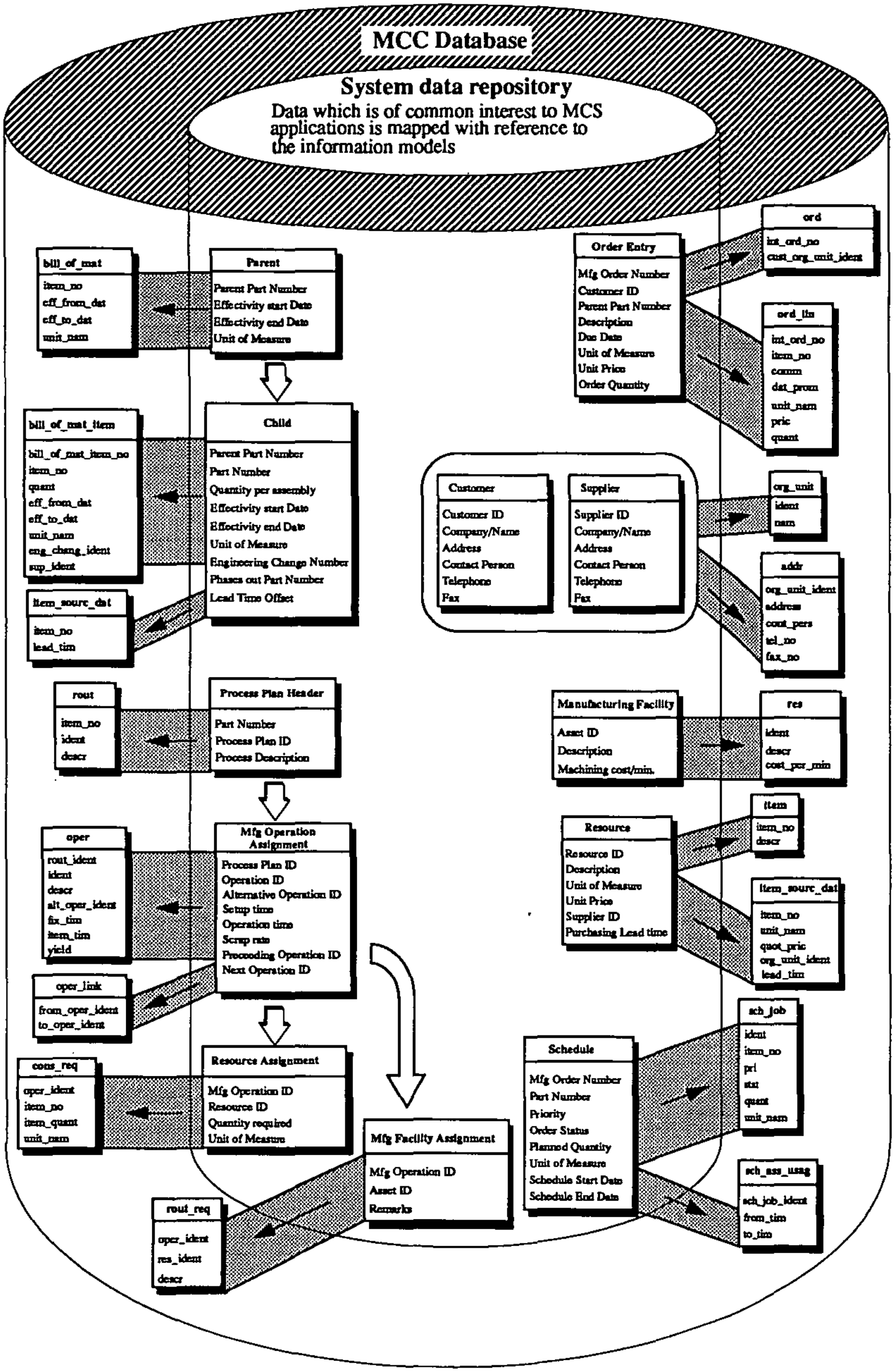


Figure 7-6 : Mapping between information entities

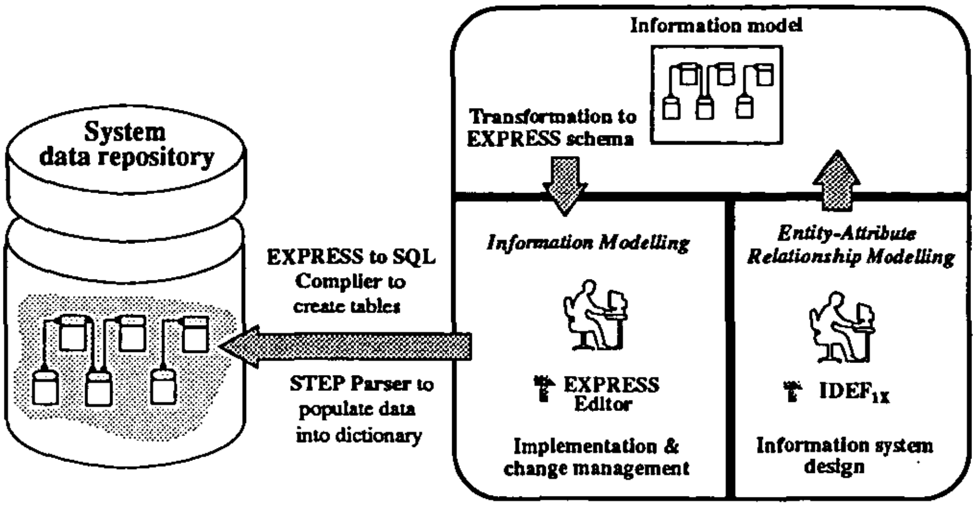

Information model
Transformation to EXPRESS schema
System data repository
EXPRESS to SQL Compiler to create tables
STEP Parser to populate data into dictionary
Information Modelling
EXPRESS Editor
Implementation & change management
Entity-Attribute Relationship Modelling
IDEF1X
Information system design

Figure 7-7 : Set of build tools to develop data repository

during activity (a) by using the $IDEF_{1X}$ modelling tool. The reader should refer to Appendix XIV for details on the EXPRESS based information model used as the global schema of the MCS demonstration system.

The EXPRESS information model is then compiled by applying the **EXPRESS to SQL Compiler** to generate SQL statements (responsible for creating relational tables which will be stored in the system data repository). In this case, the MCS demonstration system data repository was chosen to be ORACLE based. ORACLE has a similar nature to that of the MCC database, and this simplified to some extent, information sharing and transfer between the two databases. Hence the system data repository was structured according to the EXPRESS defined schemas which has the important benefit that relationships are strictly maintained, resulting in relevant information contained in the tables and inter-relationships among information objects being automatically generated and stored in a data dictionary via use of the **STEP Parser.**

(d) Populating the system data repository

At run-time the shared information (stored in the system data repository) is accessed through using services offered by a database 'driver'. This is achieved via suitably defined data access objects (as illustrated in Appendix VI), thereby making them accessible to the other MCS functional components. Thus the system data repository effectively serves as the focal point for exchanging, updating and sharing information of common concern between the various MCS functional components concerned. The services of the database 'driver' and the data access objects are required to have a knowledge of the mapping processes (defined during activity (b)) between local and global database schema as illustrated in Figure 7-6.

Using a similar principle, the data store of the cell controller was mapped onto the system data repository; this was also to establish a share information capability within the MCS demonstration system.

(II) Flexible interconnection of the functional components of the MCS

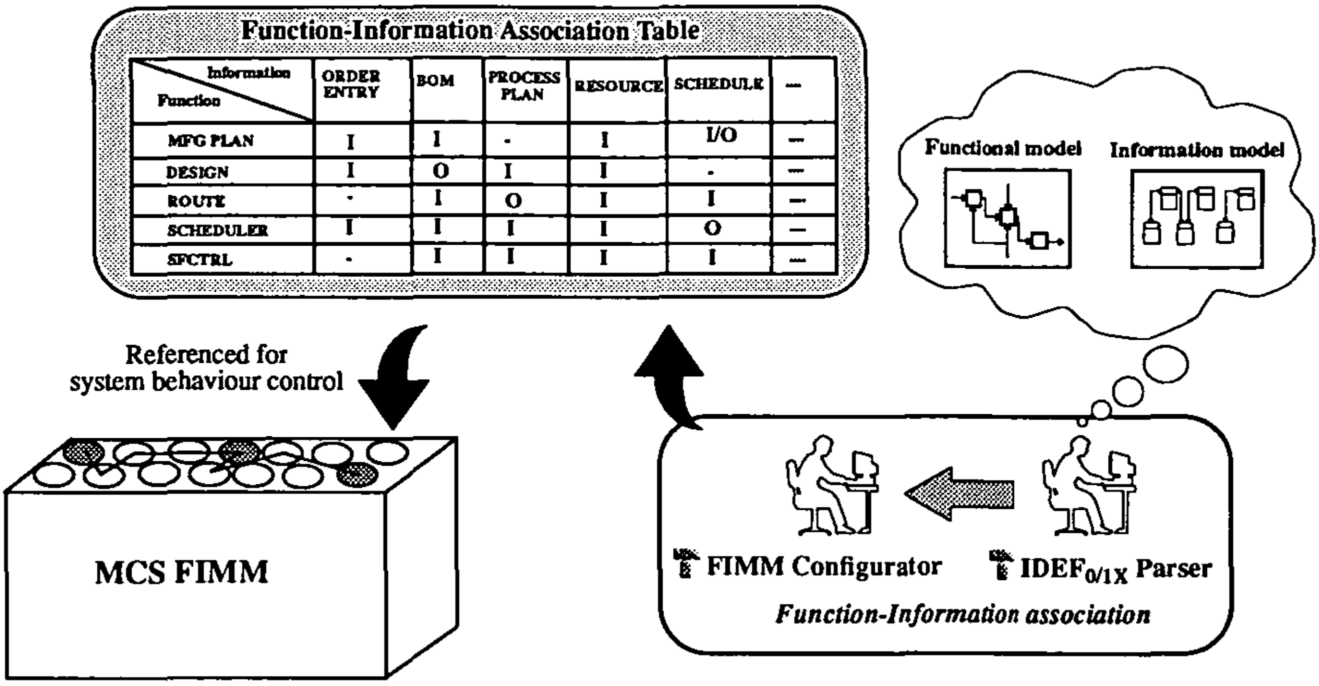
The task here is to interconnect the various MCS functional components which will normally be heterogeneous in nature and reside at different computer nodes of a distributed system. As earlier explained the CIM-BIOSYS IIS was chosen to achieve flexible interconnection among such components. However, viewed from the perspective of a CIM-BIOSYS system builder MCC is a non-conformant (or alien) application, i.e. it is not inherently compatible with the CIM-BIOSYS IIS architecture. Thus it was necessary for the author to design and implement an 'alien application shell' specially developed for MCC; this to provide it with sufficient capability to use the common integration services offered by the CIM-BIOSYS IIS (which includes inter-process communication and information management). Program listings related to the 'alien application shell' for MCC are included in Appendix XX. Conversely, being originally conceived by MSI researchers, the cell controller and the decision support system are conformant applications and they can operate directly over the CIM-BIOSYS IIS; hence they have no need of an 'alien application shell'.

The MCS functional components and the system data repository need to be registered users of the CIM-BIOSYS IIS. This registration requires their actual physical location over the local area network (LAN) to be clearly specified and stored within a CIM-BIOSYS configuration file. This knowledge is essential to the CIM-BIOSYS IIS during its normal operation in order for it to (i) activate required MCS functional component; and (ii) enable independent and transparent information access via the database 'driver', i.e. without the need for users to have detailed knowledge of integration issues.

(III) Establishment and configuration of an appropriate functional interaction capability

In this meta-step, as illustrated in Figure 7-8, the IDEF$_{0/1X}$ Parser is used to process the IDEF$_0$ function model (described in Figures 6-4, 6-5 and 6-6 in section 6.2.2) and the IDEF$_{1X}$ entity-attribute relationship model (described in Figure 7-3) as encoded by their generated reports (see Appendices XVI and XV respectively). This process can be viewed as enacting the MCS function and information models in order to establish associations between them. The reader should refer to Figure 6-7 in section 6.2.3 for an illustration of principles involved here.

Subsequently the FIMM Configurator is used to formally establish and configure associations between MCS functions and their shared information requirements, this being represented by



Function-Information Association Table

Information / Function	ORDER ENTRY	BOM	PROCESS PLAN	RESOURCE	SCHEDULE	—
MFG PLAN	I	I	-	I	I/O	—
DESIGN	I	O	I	I	-	—
ROUTE	-	I	O	I	I	—
SCHEDULER	I	I	I	I	O	—
SFCTRL	-	I	I	I	I	—

Figure 7-8 : Configuration of MCS FIMM

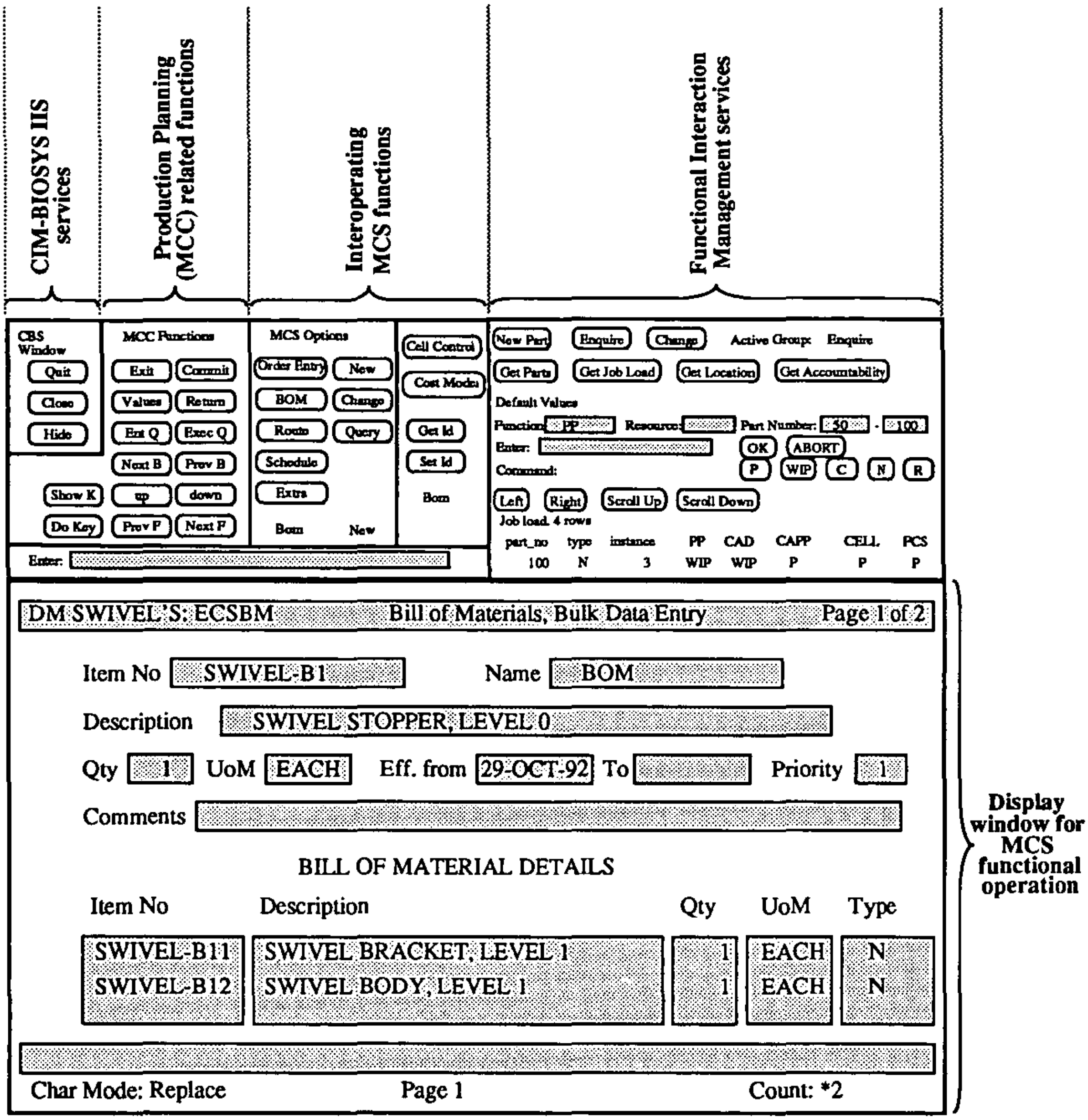
the Function-Information Association Table of the MCS FIMM. The Table is referenced by the MCS Functional Interaction Manager (during run-time) to govern the dynamic behaviour of the system in a controlled and co-ordinated manner based on predefined sequences of activities and information needs established between MCS functions.

(IV) Establishment and configuration of user interface capabilities

As illustrated in Figure 7-9 and earlier described, a configurable 'application shell' was created to act as a generic MCS front-end user interface. In the proof-of-concept MCS demonstration system this 'application shell' was configured to

- provide a consistent and simple user interface to the various MCS functional components, thereby providing an effective working environment for human interaction during run-time.
- provide human access to MCS FIMM tools, CIM-BIOSYS IIS and MCS components, in a way which facilitates system construction, management and change.

The reader should refer to Appendix XXI for further details on the services and options offered through this user interface. Details on the mechanism developed to enable communication between MCC and the 'application shell' are provided in Appendix XXII.



Active Group: Enquire

Default Values

Function: PP Resource: Part Number: 50 - 100

Enter:

Command:

Job load. 4 rows

part_no	type	instance	PP	CAD	CAPP	CELL	PCS
100	N	3	WIP	WIP	P	P	P

DM SWIVEL'S: ECSBM Bill of Materials, Bulk Data Entry Page 1 of 2

Item No SWIVEL-B1 Name BOM

Description SWIVEL STOPPER, LEVEL 0

Qty 1 UoM EACH Eff. from 29-OCT-92 To Priority 1

Comments

BILL OF MATERIAL DETAILS

Item No	Description	Qty	UoM	Type
SWIVEL-B11	SWIVEL BRACKET, LEVEL 1	1	EACH	N
SWIVEL-B12	SWIVEL BODY, LEVEL 1	1	EACH	N

Char Mode: Replace Page 1 Count: *2

Figure 7-9 : 'Application shell' configured to coalesce MCS components and integrate with MCS FIMM and CIM-BIOSYS IIS

7.2.2 Analysis and Discussion

The following sections summarise the most important observations and findings resulting from the implementation and running of the proof-of-concept MCS demonstration system :

(A) Development process

(i) *A higher entry point to integrated system development is offered which provides a structured path towards a more organised and prescribed approach.* This property is attributed to the application of generic reference and functional models which capture and encapsulate generic working knowledge of part manufacture, this in terms of clearly specified and defined information flows as well as relationships between MCS functions and their information needs. This approach avoids continuously re-inventing the wheel, which is a common disadvantage of developing custom built MCS solutions from scratch.

(ii) *It is necessary to have detailed knowledge of the proprietary information structures used in proprietary databases* (MCC in this case) *in order to understand their semantics and structure* sufficiently well enough to share information in a flexible and effective manner. Such knowledge is required before the relevant information can be accessed or mapped from the database of any "as is" MCS application onto the system data repository, thereby establishing its common usage. Thus it is vital to gain a sufficient level of support and understanding of would be interoperating software products from their manufacturer (this being provided by John Brown Systems PLC for MCC in this study). Without such knowledge there will be severe restraints on further progress in the development of interoperable systems.

(iii) When incorporating "as is" MCS applications (of the MCC ilk) into interoperating systems, *some degree of data duplication is inevitable.* However, as an underlying axiom of the methodology adopted in this research study is to decouple MCS functions from their information repositories, local changes (i.e. on the locality of a single component) to information and functional aspects of a MCS component will have minimal effect on other applications. Thus change management is much facilitated and it can be expected that functional and information needs of the system can be handled in a largely separate and less complicated manner.

(iv) The system data repository can be expected to be only partially populated (such as with information of common concern to a single data store, like the MCC database); thus *a degree of data independence is offered where the individual local databases retain their autonomy and hence can continue to serve their existing customer set.*

(v) The set of build tools used in this case study provides a more formal and structured approach to engineering interoperable manufacturing control systems solutions, as compared to others observed in the literature by the author or other MSI researchers. Thus the software toolset facilitates ease of system development and change management beyond that of alternative methodologies reported.

(B) Operation during run-time

(i) *The MCS functions (contained within interoperating components) are made available to MCS users through the configured 'application shell'*, which coalesces and integrates the interworking of the MCS functions via use of the MCS FIMM tools and the CIM-BIOSYS IIS. This can result in considerable synergy, not only in providing any single user with access to specialised MCS functions to support part manufacture but also in enabling the sharing of concurrent knowledge in a way which can support better and more informed decision making. For example the user is able to take into account information regarding availability of manufacturing resources and facilities as well as manufacturing capability when considering the requirements and specifications of parts accepted for manufacture. Thus the proof-of-concept system has been shown to promote intra-organisation integration, where an effective and defined channel for dissemination of knowledge and information is made available via a consistent and effective front user interface.

(ii) Discipline is well maintained when MCS components interact during run-time. As functional interaction is based on data availability and clearly predefined sequences of activities, any conflict or misunderstanding that might occur, in relation to information flows and needs, are effectively eliminated. For example, when the production schedule is required by the cell controller to control and monitor shop floor execution, the user issues a request through the configured 'application shell' for access of the production schedule (stored in the system data repository). The Functional Interaction Manager of the MCS FIMM would interpret the user's request and check for availability of the production schedule via its status management mechanism. The Functional Interaction Manager also ensures that the production schedule has been processed and prepared by the finite capacity scheduler in accordance with the sequence of activities predefined in the Data I/O Table, prior to triggering off the cell controller to proceed with access of the requested information from the system data repository.

Hence cooperation among interoperating software components is enabled which is not normally possible by conventional means.

Finally, it should be highlighted that the demonstration system implemented offers an effective means of improving interworking among finite capacity scheduling, cell control and production planning systems and represents a step towards more open versions of such systems which could provide even greater benefit from such a facility. *This in turn will help to narrow the wide gap which currently exist between production planning and the shop floor.*

Chapter 8

Conclusions and Recommendations

8.1 Contributions to Knowledge

The methodology devised and used in this research study supports the formal design and development of interoperable systems where the mechanisms and software toolset developed enable software components to "functionally interact" in a coherent manner by sharing information of common interest; this through accessing distributed data repositories in an efficient, highly flexible and standardised manner.

The viability of this methodology has been tested by selecting or producing software mechanisms and tools which collectively support the implementation of that methodology whilst enabling the interoperation of MCS components. By doing so, the methodology has been shown to meet the requirements identified in section 1.4, namely to enable software interoperability in an effective manner by facilitating :

- **Information Sharing**
- **Interconnection**
- **Control of System Behaviour**
- **Consistent User Interface Capabilities**
- **System Design and Development**

<u>(I) Information Sharing</u>

- *Generic reference models have been identified and defined by the author which underpin the sharing of information of common interest among production planning, product design, finite capacity scheduling and cell control systems.* They constitute prime information which are required by various components of MCS, as mentioned. *The generic reference models specified are characterised by their generalised applicability, while being sufficiently flexible to enable customisation to suit specific user needs. Indeed the reference models offer promise as being effective in addressing current problems which result from a lack of standardisation in information representation and exchange for MCS components.* The applicability and effectiveness of the generic reference models was demonstrated in a case study carried out in collaboration with UBMC. Here the database of a commercially available proprietary CAPM software package, namely ELMS, has been restructured with reference to the generic information models, this in order to facilitate the incorporation of additional functionality.

- *An information architecture has been devised and adopted which establishes structure and uniformity whilst enabling sharing and transfer of information between MCS components, this via use of the generic reference models.* The information architecture acts as a global library of information entities, providing mechanisms for representing and managing physical data, which is actually stored in a fragmented fashion within a number of heterogeneous data stores. Hence the information architecture provides a foundation for defining, identifying, and integrating both specific and generic information entities.

(II) Interconnection

- *The CIM-BIOSYS integrating infrastructure (IIS),* developed by other researchers at the MSI Research Institute at Loughborough University of Technology, *has been used to structure and simplify problems of realising interconnection among MCS components.* It separates integration and application issues in a manner which resolves differences in the physical system relating to heterogeneity, distribution and data fragmentation. The CIM-BIOSYS IIS provides low level common integration services for inter-process communication and information management. It maps distributed processes (embodied in MCS components) onto the physical data repository contained within a target manufacturing system. The CIM-BIOSYS IIS offers "soft" or flexible integration of MCS activities so as to enable their reconfiguration and incremental development.

  The author's research has contributed to the enhancement of the functionality and capability of a database 'driver' which handles data intensive activities. By doing so, this 'driver' (via the IIS) ensures consistent, reliable, transparent and open access of information stored in the data repository making it available to distributed processes in a device independent manner.

  Application of the CIM-BIOSYS IIS and the enhanced 'driver' has been demonstrated in a proof-of-concept MCS demonstration system built as part of this research study. Here interoperation of a number of typical software applications has been enabled.

(III) Control of System Behaviour

- The MCS Functional Interaction Management Module (FIMM) was designed and developed as part of this research study to (i) formally and flexibly structure threads of functionality embedded within various MCS components; and (ii) facilitate their interaction (during run-time) in a controlled, co-ordinated and deterministic manner, where considerations for associated preceeding and succeeding activities and for their shared information needs are taken into account.

The FIMM offers an effective framework for governing system behaviour in a data-driven manner based on functional dependencies and information needs and availability. The MCS FIMM can be viewed as constituting high level integration mechanisms and tools of an IIS. It builds upon the low level, general purpose integration services and tools of the CIM-BIOSYS IIS, which provide standard data inter-communication and information transfer facilities to interconnect MCS software components. The FIMM Configurator has been developed to configure the MCS FIMM in a manner which can meet specific user functional and information needs.

(IV) Consistent User Interface Capabilities

- A highly reconfigurable generic 'application shell' has been developed to serve as the front-end user interface. It provides consistent access to and coalesces the interoperation of various MCS functional activities over the CIM-BIOSYS IIS. This user interface capability supports intra-organisation integration across functional boundaries where it

  - enables access to MCS related functions which are distributed across the local area network. Hence *users are provided with a system-wide working viewpoint, this to effect better and more informed decision-making in relation to the specific tasks they perform.*
  - *makes functional interaction management easier by effectively insulating the user from the complexities and tedium involved,* which will be taken care of by the relevant services offered by MCS FIMM and the CIM-BIOSYS IIS.

  The functionality of the 'application shell' has been highlighted and demonstrated in the proof-of-concept MCS demonstration system produced in this research study.

(V) System Design and Development

- *As part of this research study a new methodology and software toolset have been developed which use models to formally structure and support implementation, run-time and change processes, this in a way which supports the various life cycle phases of systems.* In this approach, system design and modelling methods, which typically provide means of representing functional and information views of a system, serve as the entry point. The information and function models created are exploited during downstream life cycle phases to ensure clarity, consistency, accuracy and re-utilisation of knowledge and data between phases. This is enabled via a set of life cycle support tools which have been developed to closely couple IDEF$_0$ and IDEF$_{1X}$ modelling tools (used for functional and entity-attribute relationship modelling respectively) with tools based on EXPRESS (which enable information modelling). As indicated in Table 8-1, some of these tools were developed by other MSI researchers.

Life Cycle Support Tools	Developed by
EXPRESS to SQL Compiler **STEP Parser**	Other MSI researchers
IDEF$_{1X}$ to EXPRESS transformation tool **IDEF$_{0/1X}$ Parser** **FIMM Configurator**	Author as part of this research study

Table 8-1 : List of life cycle support tools developed

Accordingly, these tools reference, access, manipulate and reformat data (corresponding to function and information models) so that it assumes a form which is suitable to structure and enable various downstream life cycle processes. *Hence the software toolset offers system builders a more formalised and structured way of creating and maintaining integrated manufacturing systems, thereby catering for their need to adapt and respond to changes in system requirements.*

Use of the set of tools has been assessed by formally structuring and supporting implementation, run-time and change processes associated with the proof-of-concept MCS system produced as part of this research study.

Hence the requirement specification for enabling software interoperability (identified in section 1.4) has been met; in proof-of-concept form, it has proved possible :

- to identify and specify architectural models of system functionality and information which themselves are based on studies of the inter-dependency of functions and commonality of information shared among production planning, product design, process planning, finite capacity scheduling and cell control processes.

  This has been met in relation to the work carried out to satisfy the requirement specification in (I).

- to address key issues of managing functional interaction, i.e. to study a means of coordinating and synchronizing MCS functions. This by enabling and managing the interoperation of associated software applications in a flexible manner.

  Work done to satisfy (II), (III) and (IV) has provided a capability and working mechanism which successfully address the issues related to functional interaction and interconnection between MCS components.

- to provide a formalised and structured methodology which can cope with high levels of complexity and change, straddling design, implementation, run-time and maintenance life cycle phases of interoperable systems. This to enable overall system reconfigurability, more optimal system design and operation and a reduction in the time and effort involved in creating such systems.

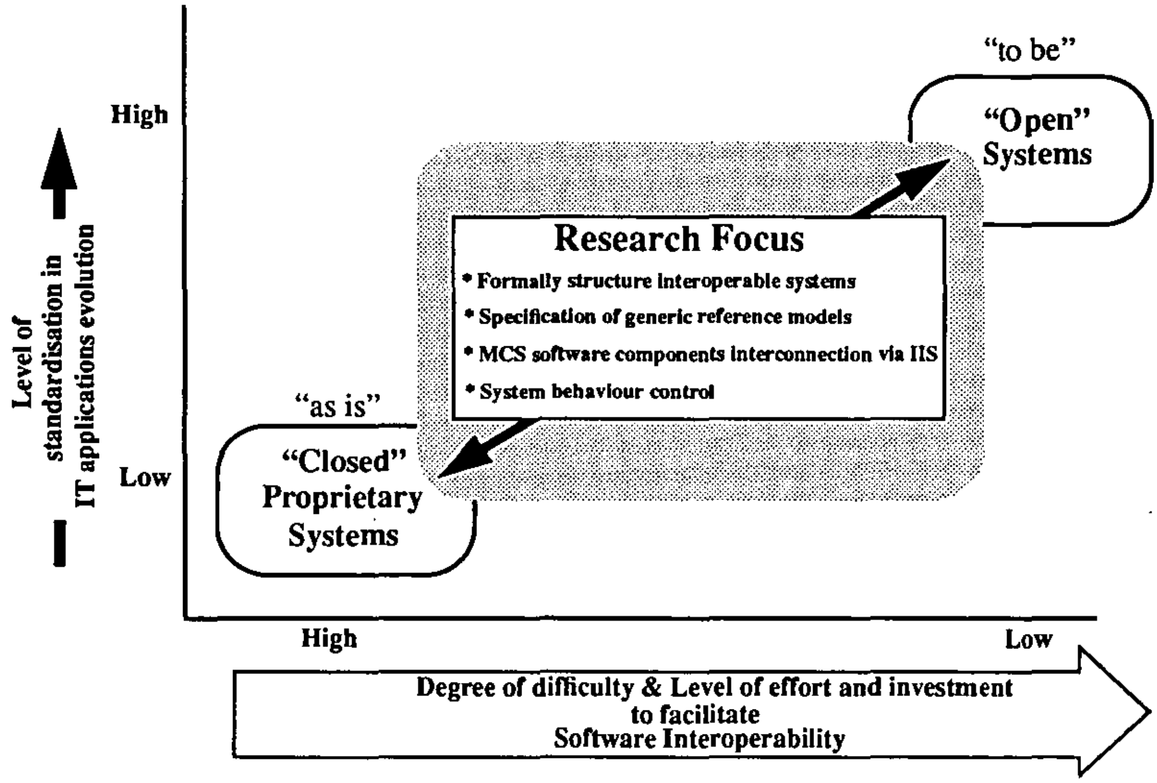
  An innovative approach which facilitates support of the MCS life cycle has been realised in order to satisfy the requirement specification in (V).

In conclusion it is evident that the realisation of a 'fully specified open standard for software interoperability', which enables on a widespread basis unconstrained interaction and interchange between heterogeneous software components, is currently an impractical goal. This is attributed to the enormous complexity of the problem and to the many outstanding standardisation issues which have yet to be resolved. Much of it depends upon

- availability of suitable MCS with facilities required for the following which should be sufficiently standard to make many people adopt and write to those standards :
  - interprocess communication
  - information sharing and management
  - interconnection
- availability of acceptable reference models which can describe in a comprehensive manner (i) information flow and requirement; (ii) functional activities and their inter-relationships and dependencies; and (iii) system behaviour.
- availability of integrated life cycle support systems to design better MCS solutions.
- progressive release of more interoperable MCS components (with more modular and atomic functionality) from vendors.

However, on the other hand, the use of contemporary 'turnkey' or 'custom built' solutions (which do not adhere to any open standards) is not tenable for systems requiring interoperation of many MCS components. Constraints in software interoperability will undoubtedly remain if such proprietary systems are not designed to enable access to their threads of functionality or underlying information.

Bearing in mind these difficulties, ***this research has offered a realistic approach which can be considered to be part-way between the extremes of "open" and "closed" systems,*** as illustrated in Figure 8-1. The emphasis has been to provide a means of enabling a degree of software interoperability which overcomes (a) limitations inherent in contemporary MCS components and solutions; and (b) associated and inherited problems concerned with achieving their interoperation. Hence based upon the implementation studies in this research, an infrastructure that enables a degree of interoperation is offered which not only allows for the adoption of legacy (or "as is") MCS components but also the introduction of a new generation of highly reconfigurable, modular and more open MCS software products, i.e. "to be" products (as they become available to industry). Thus a migration path is offered towards "open" manufacturing control systems which are readily adaptable in the face of changing functional and operational requirements.



Level of standardisation in IT applications evolution
High
Low
"to be"
"Open" Systems
Research Focus
• Formally structure interoperable systems
• Specification of generic reference models
• MCS software components interconnection via IIS
• System behaviour control
"as is"
"Closed" Proprietary Systems
High
Low
Degree of difficulty & Level of effort and investment to facilitate Software Interoperability

Figure 8-1 : Degree of Software Interoperability

8.2 Further Recommendations

As part of the immediate requirement for progressive enhancement of the software interoperability methodology described in this thesis, the author recommends further investigation and development in the following key areas in order to address existing deficiencies :

(A) Integration and management of multi-vendor database systems

A number of database "drivers" have been created to enable access to the Progress, Ingres and ORACLE RDBMS. However, *the 'drivers' are restricted and dedicated to addressing the idiosyncrasies of the specific database system they service. They are not equipped to handle semantic differences among different database systems.* Currently, collating data from different database systems can be somewhat tedious where the data needs to be filtered through a series of programs in order to resolve semantic differences and to convert them to the required format when retrieving and populating data across different database systems.

Thus further research is required towards developing a more elegant solution for global data management and manipulation across such multi-vendor database systems. The emphasis should be to provide a means of unifying data across different database systems, resolving the heterogeneity and semantic differences between them and presenting the information in accordance with the format requested by the user. Hence information would appear as part of a unified system-wide database.

(B) Modelling and simulation capability for system behaviour control

The MCS Functional Interaction Management Module (FIMM) was developed to provide an effective framework which (i) formally and flexibly structures threads of functionality embedded within the various MCS components, and (ii) facilitates their interaction (during run-time) in a controlled, co-ordinated and data-driven manner.

It would be beneficial to also provide a modelling and simulation capability which during initial system design or prior to making a system change, enables performance analysis of the interoperable system. This would help identify and enable a study of conditions under which the system can interoperate in a more optimal manner in relation to the constraints imposed. The following could be considered in the analysis for system behaviour control :

- functional dependencies, this to properly structure the co-ordination of the combined activities in order to clearly identify succeeding and preceeding activities and their shared information requirement.

- type and nature of suitable associations (such as one-to-one, one-to-many, many-to-many, etc.) to be established between functions and their information needs so as to ensure better information flow for enhanced system performance. This would help alleviate conflicts, contentions and ambiguities with regard to information requirement in the interoperating system.

Such a capability could be provided for in the form of a software tool, coupled closely with the FIMM database to access and use the relevant data needed.

Finally, the research methodology described in this thesis can only facilitate software interoperability which is sufficiently useful and effective in satisfying mid-term needs. In order to provide for more effective and universally applicable interoperable solutions, the author expects future interoperability development in the following major areas :

- ***Advancement of life cycle support approaches leading to simulation, emulation and execution of MCS so as to result in better designed systems.***
- ***Standardisation of information reference models and functional models.***
- ***Improved and standard infrastructural facilities.***
- ***Progressive development of a new generation of more open MCS components (possibly on the base of general software developments).***

PUBLICATIONS

The following refereed technical publications have been made in relation to this research study :

I) International Journals

- Singh, V., Weston, R. H., 1994,
  Functional interaction management : A requirement for software interoperability
  Journal of Engineering Manufacture, The Institution of Mechanical Engineers, Part B,

- Singh, V., Weston, R. H., 1994,
  Life Cycle Support of Manufacturing Systems based on an Integration of Tools
  Journal of Production Research

- Singh, V., Weston, R. H., 1994,
  Information Models : A Precursor to Software Interoperability
  Journal of Production Planning and Control

- Weston, R. H., Singh, V., 1994,
  Structured Specification and Construction of Open Manufacturing Control Systems
  Journal of Manufacturing Systems
  SME (Society of Manufacturing Engineers, USA)

II) Proccedings of International Conferences

- Singh, V., Weston, R.H., May 1994,
  Software Interoperability for Integrated Manufacturing,
  A Reference Model Driven Approach
  Conference on Data and Knowledge Systems for Manufacturing and Engineering
  (DKSME '94), Hong Kong.

- Singh, V., Weston, R. H., September 1993,
  New Generation of "Open" Manufacturing Control Systems
  for "Seamless" Integration in CIM
  Conference on Computer Integrated Manufacturing (ICCIM' 93), Singapore

REFERENCES

Afferson, M., Andrews, J. K., Muhlemann, A. P., Price, D. H. R., Sharp, J. A., 1992,
Generic manufacturing information systems development via template prototyping,
European Journal of Information Systems, Vol. 1, p379-386

Aguiar, M.W., Weston, R.H., 1993a,
CIM-OSA and stochastic time Petri nets for behavioural modelling
and model handling in CIM systems design and building,
Procs. of the Institution of Mechanical Engineers, Vol. 207. Part B. Journal of Engineering Manufacture, pp147-158.

Aguiar, M. W., Weston, R. H., August 1993b,
Reference Architectures for Enterprise Integration,
Procs. of CARS/FOF' 93 Conference, USA

Akif, H. C., Documeings, G., 1991,
Computer aided GRAI method (C. A. GRAI),
Procs. of Advances in Production Management Systems - IFIP/91, France, pp283-292

AMR (Advanced Manufacturing Research), March 1991,
Application Enabler,
Report, USA

Anscombe, J., November 1992,
Integration - Breaking the Barriers to Excellence,
Procs. Twenty-seventh Annual BPICS Conference, Birmingham, UK, pp89-103

Arngrimsson, G., Vesterager, J., August 1992,
STEP : Experiences from actual use of the standard,
Procs. of IFIP Working Group 5.7, Conference on Integration in Production Management Systems, Eindhoven, Netherlands, pp23-35

Bailin, S. C., 1989,
An object-oriented requirements specification method,
Communications of the ACM, Vol. 32, No. 5, pp608-623

Barkmeyer, E. J., 1989,
Some Interactions of information and control in Integrated Automation systems,
Advanced Information Technology, Industrial Material Flow Systems, Springer-Verlag

Batini, C., Lenzerini, M., Navathe, S. B., December 1986,
A Comparative Analysis of Methodologies for Database Schema Integration,
ACM Computing Surveys, Vol. 18, No. 4, pp322-364

Bauer, A., 1991,
Shop floor control systems : from design to implementation
Chapman and Hall

Beech, D., Ozbutun, C., 1990,
Object databases as generalizations of relational databases,
Proc. of the Object-Oriented Database Task Force Group Workshop,
Ottawa, Canada, pp119-135

Beerit, C., October 1993,
New Directions in Database Management Systems
Procs. of Fifth Jerusalem Conference on Information Technology, Israel, pp500-506

Bohse, M. E., Harhalakis, G., 1987,
Integrating CAD and MRP II Systems,
CIM Review, Vol. 3, No. 4, pp7-15

Bond, T. C., 1993,
An investigation into the use of OPT production scheduling,
International Journal Production Planning and Control, Vol. 4, No. 4, pp399-406

Breitbart, A., Morales, H., Silberschatz, A., Thompson, G., October 1993,
Multidatabase Concurrency Problems
- Multidatabase Transctions Concurrency Control Mechanisms,
Procs. of Fifth Jerusalem Conference on Information Technology, Israel, pp507-519

Bright, M. W., Hurson, A. R., Pakzad, S. H., 1992,
A Taxonomy and Current Issues in Multidatabase Systems,
IEEE, pp50-59

Buchman, A. P., 1984,
Current Trends in CAD Databases,
Computer-Aided Design, Vol. 16, No. 3

Buyer's Guide Supplement, February 1990,
Production Management Software, Industrial Computing, pp46-58

Chang, Tien-Chien, 1985
An Introduction to Automated Process Planning Systems
Prentice-Hall

Chaudhri, A. B., Revell, N., 1994,
Object database benchmarks: past, present and future,
Proc. of Object-Oriented Databases: Realising their Potential and Interoperability with RDBMS, London, UK

Chaudhri, A. B., 1993
Object database management systems: an overview
BCS OOPS Newsletter, No. 8, pp6-15

CIM-OSA ESPRIT Consortium AMICE, 1989,
Open System Architecture for CIM,
Springer-Verlag, Berlin (D)

CIM Strategies, March 1990,
DELTA factory floor manager combines data management methods, pp7-10

CIM Strategies, March 1991,
Application Case Study, Interoperability standards form a base for CIM,
Vol. 8, No. 3, pp4-7

Clements, P., Coutts, I.A., Weston, R.H., September 1993,
A life-cycle support environment comprising open systems manufacturing modelling methods and the CIM-BIOSYS infrastructural tools,
Proc. of the Symposium on Manufacturing Applicaton Programming Language Environment (MAPLE) Conference, Ottawa, Canada, pp181-195.

Clements, P., October 1992,
The application of EXPRESS modelling and tools within an integration platform,
Second EXPRESS Users Group, Dallas, USA

Clements, P., Hodgson, A., Leech, M., Ryan, A., November 1991,
Information Systems Modelling and Implementation in an industrial environment,
Procs. of AUTOFACT '91, Chicago, Illinois, USA

Clements, P., March 1991a,
Internal Report on the STEP Parser, Loughborough University of Technology

Clements, P., February 1991b,
Internal Report on the EXPRESS to SQL Compiler, Loughborough University of Technology

Codd, E. F., 1992,
Dr. Codd on "End of Relational",
DBMS, Vol. 5, No 11:6

Colquhoun, G. J., Baines, R. W., Crossley, R., 1993,
A state of the art review of IDEF0,
International Journal Computer Integrated Manufacturing, Vol. 6, No. 4, pp252-264

Computing, November 1991,
Database giants revamp products, p11

Cutts, G., 1991,
SSADM Structured Systems Analysis and Design Methodology,
Blackwell Scientific, Oxford, UK

Czernik, S., Quint, W., 1992,
Selection of methods, techniques and tools for system analysis
and for the integration of CIM elements in existing manufacturing organizations,
International Journal Production Planning and Control, Vol. 3, Part 2, pp202-209

DATAPRO, March 1992,
Manufacturing Automation Series : Factory Automation Systems,
McGraw Hill , USA

Date, J. , 1986,
An Introduction to Database systems,
Vol. 1, Addison-Wesley Publishing Co, Inc

Davis, G. B., Olson, M. H., 1987,
Management Information Systems,
Second Edition, McGraw-Hill, pp502-504

De Toni, A., Caputo, C., Vinelli, A., 1988,
Production management techniques,
International Journal of Operations and Production Management, Vol. 8, No. 2, pp35-51

De Vaan, M. J., July-September 1992,
Introduction MRP II, with enhancements: the case of a furniture manufacturer,
Intrnational Journal Production Planning and Control, Vol. 3, No. 3, pp258-263

Dinitz, M., July 1990,
Configure-To-Order: An industry challenge,
Industrial Engineering, pp21-22

Drucker, P. F., November 1991,
The Factory of the Future,
World Executive's Digest, pp26-32

DTI, 1993,
Computer Integrated Manufacturing - A Survey of Worldwide R & D

DTI, 1989,
PA Consulting Group, *Manufacturing into the late 1990s,* HMSO

DTI, 1987,
UK, Through MAP to CIM, Moore & Matthes Ltd

ELMS Technical Manual, 1990

ESPRIT Consortium, 1989,
Open System Architecture for CIM,
Project 688, Vol. 1, Springer-Verlag, pp13-16

Evans, C. D., Meek, B. L., Walker, R. S., 1993,
User Needs in Information Technology Standards,
Butterworth-Heinemann Ltd (Publisher), UK

Foong, N. F., Ang, K. P., Singh, V., May 1992,
Computer Simulation as a Tool for Integrated Manufacturing,
Procs. Asia-Pacific Industrial Automation (IA)' 92 Conference, Singapore

Fritsch, C. A., 1989,
Information Dynamics for Computer Integrated Product Realisation,
NATO ASI Series, Springer Verlag, Vol. F53:, pp21-38

Fry, T., Karwan, K., Baker, W., 1993,
Performance measurement systems and time-based manufacturing,
International Journal of Production Planning and Control, Vol. 4, No. 2, pp102-111

Golberg, C. J., Winter 1993,
Object Oriented Databases - The New Wave in RDBMS Technology,
ORACLE, Vol. VII, No. 1, pp35-39

Goldratt, M., E., 1988,
Computerized shop floor scheduling,
International Journal of Production Research, Vol. 26, No. 3, pp443-455

Gould, L., August 1992,
CIM Interface Modules : A route to Open Systems,
Managing Automation, Vol. 7, No. 8, pp47-50

Goyal, S. K., Gunasekaran, T. Martikainen, Yli-Olli, P., 1993,
Design of optimal configuration for a multi-stage production system,
International Journal of Production Planning and Control, Vol. 4, No. 3, pp239-252

Halevi, G., Weil, R., 1992,
CAPP as concurrent link between Design and Production Management,
IFIP Transactions Part B Applications in Technology, Vol. 6, pp177-184

Halladay, S., Wiebel, M., 1993
Object-Oriented Software Engineering,
Lawrence, Kan.: R & D Publications

Halsall, D. N., Muhlemann, A. P., Price, D. H. R., September 1993,
A Production Planning and Resource Scheduling Model from Small Manufacturing Enterprises,
Procs. Ninth National Conference on Manufacturing Research

Harhalakis, G., Lin, C. P, H. Hillion, Moy, K. Y., 1990,
Development of a factory Level CIM Model,
Journal of Manufacturing Systems, Vol. 9, No. 2 , pp116-128

Hars, A., Heib, R., Kruse, Chr., Michely, J., Scheer, A., -W., May 1992,
Reference Models for Data Engineering in CIM,
Procs. Eighth CIM-Europe Annual Conference, Birmingham, UK, pp249-260

Hars, A., 1990,
CIDAM - modules for the creation of CIM,
Procs. Sixth CIM-Europe Annual Conference, pp286-295

Hayes, F., Spring 1992,
Esperanto for Databases,
Unixworld-Supplement : Special Report Interoperability, pp49-51

Himes, D. A., 1993,
Database Interoperability and portability through standards,
Procs. of the Second International Conference on Parallel and Distributed Information Systems, pp225-256

Hind, C. J., West, A. A., Williams, D., J., 1990,
The use of object orientation for the design and implementation of manufacturing process control systems,
Internal Report, Dept of Manufacturing Engineering, Loughborough University of Technology, LUT Press

Hodgson, A., 1993,
Production Planning and Control within a CIM environment : some current developments and requirements for the future,
International Journal Production Planning and Control, Vol. 4, No. 4, pp296-303

Hodgson, A., Weston, R. H.,; 1993,
Application and Information Support Systems for Planning and Control in CIM,
ACME Review Final Report, Grant No. GR/F 69192

Hodgson, A., Waterlow, G., 1992,
Special feature: Computer-aided production management,
Computing & Control Engineering Journal, Vol. 3, No. 2, Published by IEE, ISSN 0956-3385

Hodgson, A., Weston, R. W., Sumpter, C. M., Gascoigne, A., August-September 1988,
Planning And Control Information flow in CIM,
Procs. International Conference on Factory 2000 - Integrating Information and Material Flow, Cambridge, UK, pp49-56

Higgins, P., Tierney, K., Browne, J., September 1991,
Production Management State of the Art and Perspectives,
Procs. Fourth International IFIP TC5 Conference, Computer Applications in Production and Engineering, Bordeaux, France, pp3-14

Hollyman, B., Anderson, L., January 1991,
Implementing an Open Systems Architecture,
CommUNIXations, Published by Uniforum (International Association of Unix Systems Users), Vol. XI, No. 1, pp23-29

Hughes, D., August-September 1988,
Criteria for the distribution of information processing in factory 2000,
Procs. International Conference on Factory 2000 - Integrating Information and material flow, Cambridge, England, pp45-48

ICAM, December 1985,
Information Modelling Manual IDEF1 - Extended,
ICAM Project Report (Priority 6201), D. Appleton company, Inc, Manhattan Beach, California

Ingres, 1991,
Database set of manuals Version 6.4

ISO, 1993, ISO DIS 10303-1,
Product Data Representation and Exchange Part 1 : Overview and Fundamental Principles,
International Organization for Standardization, Geneva

ISO, 1991,
MANDATE,
ISO TC184/SC4/WG8 Document N1, ISO TC184/SC4 Secretariat, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA

ITAP Technology Seminar, 1990,
Advances in Computer Integrated Manufacturing,
ITAP Technology Report No. 5/90, National Computer Board (Singapore)

Jain, K. H., Bu-Hulaiga, I. M., Summer 1992,
E-R Approach to Distributed Heterogeneous Database Systems for Integrated Manufacturing,
Journal of Database Administration, Vol. 3, Part 3, pp21-29

Jeng, B. C., Chao, W. S., July 1992,
Communicating Objects for System Integration modelling, Procs. Second International Conference on Automation Technology, Taipei, Taiwan, Vol. 2, pp307-312

Jochem, R., 1989,
An object oriented analysis and design methodology for computer integrated manufacturing systems,
Tools 89, pp75-84

Jones, G., Roberts, M., 1990,
Optimized Production Technology (OPT),
IFS Publications, UK

Joris, S. M., Vergeest, Matthijis, Sepers, June 1993,
Techniques to make CAD/CAM Systems communicative,
Procs. of the Third International Flexible Automation and Integrated Manufacturing, University of Limerick, Ireland, pp255-266

Jorysz, H. R., Vernadat, F. B., 1990,
CIM-OSA part 1 : Total enterprise modelling and function view,
International Journal Computer Integrated Manufacturing, Vol. 3, Nos. 3 and 4, pp144-156

Kaul, M., Drosten, K., Neubold, E. J., 1989,
View System: Integrating Heterogeneous information bases by object-oriented views.
Procs. IEEE International Conference on Data Engineering

Khoshafian, S., Blumer, R., Abnous, R., 1990
Inheritance and generalization in Intelligent SQL,
Proc. of the Object-Oriented Database Task Force Group Workshop,
Ottawa, Canada, pp103-118

Kochhar, A. K., Monniott, J. P., Price, D. H. R, Rhodes, D. J., Towill, D. R., Waterlow, J. G., 1987,
A study of computer aided production management in UK batch manufacturing,
International Journal of Operations and Production Management, Vol. 7, pp7-57

Koriba, M.., 1983,
Database Systems : Their Applications to CAD Software Design,
Computer-Aided Design, Vol. 15, No. 5

Kosanke, K., 1991,
Open Systems Architecture for CIM (CIM-OSA) Standards for Manufacturing,
Procs. International Conference on Computer Integrated Manufacturing (ICCIM' 91),
Singapore

Krishnamurthy, R., Litwin, W., Kent, W., April 1991,
Interoperability of Heterogeneous Databases with schematic discrepancies,
Procs. First International Workshop on Interoperability in Multidatabase systems, Kyoto,
Japan, pp144-151

Lang-Lendroff, G., Unterburg, J, June 1989,
Changes in understanding of CAD/CAM : a database-oriented approach,
Computer Aided Design, Vol. 21, No. 5, pp309-314

Lars, D. T., 1990,
Is there a "GAP" of knowledge between R&D and Production?,
Advances in production management systems, Procs. Fourth International IFIP Conference
TC5/WG 5-7, Espoo, Finland

Larsen, N. E., Alting, L., 1993,
Criteria for selecting a production control philosophy,
International Journal Production Planning and Control, Vol. 4, No. 1, pp54-68

Lee, C. Y., 1993,
A Recent Development of the Integrated Manufacturing System : A Hybrid of MRP and JIT,
International Journal of Operations and Production Management, Vol. 13, No. 4, pp3-17

Leech, M., J., March 1993,
Internal Report on CIM-BIOSYS Datastore Driver Guide, Loughborough University of Technology

Lim, B. S., July-October 1992,
CIMIDES - A Computer Integrated Manufacturing Information and Data Exchange System,
International Journal of Computer Intergrated Manufacturing, Vol. 5, No. 4 & 5, pp240-254

Logan, F. A., March 1986,
Evolutionary Cycle of an Expert CAPP System,
Procs. Conference CIMTECH, Boston, Massachusetts

Lopes, P. F., 1992,
CIM II : The Integrated Manufacturing Enterprise,
Industrial Engineering, Vol. 24, No. 11, pp43-45

Luscombe, M., 1991
Design and Implementation of Integrated Production Control systems,
Integrated Manufacturing System, Vol. 2, No. 4, pp4-8

Maier, D., 1989,
Object-Oriented Concepts, Databases, and Applications,
Edited by W. Kim and F. H. Lochovsky,
Addision-Wesley

Maji, R. K., October 1988,
Tools for development of Information Systems in CIM,
Advanced Manufacturing Engineering, Vol. 1, pp26-34

Martin, J., 1980,
Computer Data Base Organization,
Second Edition, Prentice-Hall, Englewood Cliffs, New Jersey

Martin, J., 1988,
CIM : What the Future Holds?,
Manufacturing Engineering

Maude. T., Willis, G., 1991,
Rapid Prototyping,
Pitman Publishing, London, UK

Maull, R. S., Childe, S. J., 1993,
A step-by-step guide to the identification of an appropriate computer-aided production management system,
International Journal of Production Planning and Control, Vol. 4, No. 1, pp69-76

Mayer, R. J., Painter, M. K., 1991,
Roadmap for enterprise integration,
Procs. of Autofact 91, USA

MCC Technical Manual, 1989
John Brown Systems PLC

Meta Software, 1990,
Design/IDEF User's Manual, Meta Software

Metz, S., August 1990,
Making Manufacturing Better, not just faster,
Managing Automation

Moerman, P.A., 1991,
The evaluation of technology in relation to products and markets: observations, considerations, experience, and solutions,
International Journal of Computer Integrated Manufacturing, Vol. 4, No. 1, pp2-15.

Motro, A., July 1987,
Superviews : Virtual Integration of Multiple Databases,
IEEE Trans. Software Engineering, Vol. 13, No. 7, pp785-798

Muhlemann, A. P., Price, D. H. R., Sharp, J. A., Afferson, M., 1991,
Fourth Generation languages and integrated information systems for small manufacturing companies,
International Journal Computer Integrated Manufacturing, Vol. 4, No. 1, pp16-22

Muhlemann, A. P., Price, D. H. R., Sharp, J. A., Afferson, M., Andrews, J. K., 1990,
Information systems for use by production managers in smaller manufacturing enterprises,
Procs. of the Institution of Mechanical Engineers (Part B), Vol. 204, p191-196

Olle, T. W., 1978,
The CODASYL Approach to Database Management Systems,
John Wiley and Sons, New York

ORACLE RDBMS, June 1991,
Utilities User's Guide,
Version 6.0, Oracle Corporation

ORACLE, 1992
Database set of manuals
Version 6.0, Oracle Corporation

Orr, K., Gane, C., Yourdon, E., Chen, P. P., Constantine, L. L., April 1989,
Methodology : The Experts Speak,
Byte, pp221-244

Paranuk, H. V. D., 1988,
Chapter 5 : Factory communication system, Artificial Intelligence : Implications for CIM,
IFS Publications Ltd, Springer-Verlag

Peters, T., 1989,
Thriving on Chaos,
Pan Books Ltd, UK

Perkovic, P., Spring 1991,
SQL Access and ANSI/ISO SQL and X/Open ,
COMPCON, pp120-122

Pheasey, D., November 1992,
Competitive Manufacturing - 'A Vision of the year 2001',
Procs. Twenty-seventh Annual BPICS Conference, Birmingham, UK, pp23-31

Plenert, G., 1993,
An Overview of JIT,
International Journal of Advanced Manufacturing Technology, Vol. 8, pp91-95

Preece, J., 1993,
A Guide to Usability,
Addison-Wesley

Progress, 1990,
Database set of manuals version 6.2

Ptak, C. A., 1991,
MRP, MRP II, OPT, JIT, and CIM - Succession, Evolution, or necessary combination?,
Production and Inventory Management Journal, Vol. 32, Part 2, pp7-11

Pugh, D. S., Hickson, D., J., 1989
Writers on Organization
Penguin Books (Fourth Edition), pp90-93

Rembold, U., Nnaji B. O., Storr, A., 1993
CIM
Addison-Wesley, UK

Ross, D. T., 1977,
Structured Analysis (SA) : A language for Communicating Ideas,
IEEE Transactions on Software Reliability, Vol. 3, No. 1

Rui, A., 1989,
Information support systems for the distributed planning and control in batch manufacture,
PhD Thesis. (Supervised by Weston, R.H. and Hodgson, A.): PhD awarded 1989.

Rumbaugh, J., et al., 1991
Object-Oriented Modeling and Design,
Prentice Hall International

Rusinkiewicz, M., Czejdo, B., 1987,
An approach to query processing in federated database systems.
Procs. Hawaii International Conference on Systems Sciences

Sanders, L., Mayer, R. J., Browne, D. C., Menzel, C., 1991,
Containers objects : a description based knowledge representation scheme,
Procs. of Autofact' 91, USA, pp7.39-7.50

Savolainen, T., 1991,
CIMVIEW : a tool for symbolic top-down simulation for CIM,
Procs. of Advances in Production Management Systems, IFIP, Holland

Saxe, K., November 1985,
MRP II Into CIM : The Interface Phase,
Procs. Conference Autofact '85, Detroit, Michigan

Scheer, A.-W., 1991,
CIM - Towards the Factory of the Future,
Second Edition, (Springer-Verlag)

Scheer, A.,-W., 1989,
Enterprise-Wide Data Modelling - Information Systems in Industry,
Springer-Verlag, pp259

Scheer, A.-W., 1988,
Computer Integrated Manufacturing - Computer Steered Industry
First Edition, (Springer-Verlag)

Schenck, D., December 1989,
Information Modelling Language EXPRESS,
ISO TC184/SC4/WG1 N442

Schiel, U., Mistrik, I., 1990,
Using object-oriented analysis and design for integrated systems,
Procs. of the First International Conference on Systems Integration, USA, pp125-134

Schnur, J. A., Summer 1987,
Can there be CIM Without MRP II?,
CIM Review

Schonewolf, W., Langendoen, M., Gransier, T., Baisch, R., Drossopoulos, May 1992,
Application of CIM-OSA in Machine Tool Manufacturing and Aluminium Casting,
Procs. Eighth Annual CIM-Europe Conference, Birmingham, UK, pp217-229

Shaharoun, A. M., Hodgson, A., Weston, R. H., August 1992,
Cost modelling in Advanced Manufacturing Systems,
Procs. of International Conference for Manufacturing Automation (ICMA), Hong Kong

Shunk, D., Sullivan, B., Cahill, J., Fall 1986,
Making the Most of IDEF Modeling - The Triple Diagonal Concept,
CIM Review, pp2-17

SI (Systems Integration) Group (LUT), February 1994,
Model Driven CIM : The design, implementation and management of Open CIM systems
Loughborough University of Technology
SERC/ACME Review Report No. 2, Grant No. GR/H/22798

SIM, 1993,
User and Technical Manual,
MSPL Ltd.

Singh, V., Weston, R. H., 1994a,
Functional interaction management : A requirement for software interoperability,
Procs. of the Institution of Mechanical Engineers, Part B, Journal of Engineering Manufacture

Singh, V., Weston, R.H., May 1994b,
Software Interoperability for Integrated Manufacturing, A Reference Model Driven Approach,
International Conference on Data and Knowledge Systems for Manufuring and Engineering (DKSME '94), Hong Kong.

Singh, V., Weston, R. H., September 1993,
New Generation of "Open" Manufacturing Control Systems for "Seamless" Integration in CIM,
Procs. International Conference on Computer Integrated Manufacturing (ICCIM' 93), Singapore, pp309-321

Singh, V., May 1992,
Flexible Materials Handling and Storage System for Integrated Manufacture,
Procs. Asia-Pacific Industrial Automation (IA)' 92 Conference, Singapore, pp10-21

Singh, V., October 1991,
CIM Model for Metal Machining Trade - Translating Vision into Reality,
Procs. International Conference on Computer Integrated Manufacturing (ICCIM' 91),
Singapore, pp336-341

Solberg, J. J., 1989,
Managing Information Complexity in Material Flow Systems,
NATO ASI Series, Springer Verlag, Vol. F53:, pp3-20

Ssemakula, M. E., 1987,
The role of process planning in the integration of CAD/CAM systems,
Procs. of Fourth European Conference on Automated Manufacturing (AUTOMAN 4),
Birmingham, UK

St. Charles, D. P., October 1987,
The Fractured CIM Market,
Managing Automation

Struedel, H. J., Desruelle, P., 1992,
Manufacturing in the Nineties,
Van Nostrand, Reinhold, New York

Taylor, F. W., 1947
Scientific Management
Harper and Row (Publishers)

Taylor, R. W., Frank, R. L., 1976,
CODASYL data base management systems,
ACM Computing Surveys, New York, Vol. 8, No. 1

Terry, W. R., Matz, T. W., 1989,
An object-oriented programming paradigm for synchronous manufacturing,
Computers Industrial Engineering, Vol. 17, Nos. 1-4, pp124-129

Thompson, G. R., Gomer, T., Chung, C., Barkmeyer, E., Carter, F., Templeton, M., Fox, S.,
Hartman, B., September 1990,
Heterogeneous Distributed Databases Systems for Production Use,
ACM Computing Surveys, Vol. 22, No. 3, pp237-265

Timon, F., Jagdev, H. S., Browne, J., 1990,
The Analysis of and the selection Criterion for Production Management Packages,
Advances in production management systems, Procs. Fourth International IFIP Conference
TC5/WG 5-7, Espoo, Finland, pp427-438

Van der Lans, R. F., 1989,
The SQL standard, ,
Prentice Hall

Van Donselaar, K., July-September 1992,
The use of MRP and LRP in a stochastic environment,
International Journal Production Planning & Control, Vol. 3, No. 3, pp239-246

Vollmann, T. E., Berry, W. L., Whybark, D. C., 1988,
Manufacturing Planning and Control Systems,
Dow Jones Irwin, Homewood, IL.

Waterlow, J. G., Monniott, J. P., 1986,
A study of the state of the Art in Computer-Aided Production Management in UK industry,
ACME Report

Weber, D. M., Moodie, C. L., 1989,
Distributed intelligent information systems for automated integrated manufacturing systems,
Advanced Information Technologies for Industrial Material Flow systems, Springer-Verlag

Weinberg, J. C., 1989,
Linking the CIM Plan with Operations Strategy,
Procs. Conference Autofact' 89, Detroit, Michigan

Welz, F., March 1993,
Software Interoperability within Manufacturing Control Systems,
Graduate Dissertation (Dept of Manufacturing Engineering, Loughborough University of Technology), LUT Press

Weston, R.H., 1993,
Steps Towards Enterprise-Wide Integration: a Definition of Need
and First Generation Open Solutions,
International Journal of Production Research, Vol. 31, No. 9, pp2235-2254.

Weston, R. H., Zhang, P., Murgatroyd, I. S., Coutts, I. A. and Hodgson, A., September 1991,
Soft Integrated Assembly Systems,
Procs. Fourth World Conference on Robotics Research, Pittsburg, USA, pp410-419

Weston, R. H., Gascoigne, J. D., Rui, A., Hodgson, A., Sumpter, C. M. and Coutts, I., 1988
Steps towards information integration in manufacturing,
International Journal Computer Integrated Manufacturing, Vol. 1, No. 3, pp140

Weymont, N., P., Honeyager, J. S., 1987,
Developing a CIM Architecture,
Procs. of the Digital Equipment Computer Users Society, USA

White, C. J., Winter 1993/1992,
Interoperability : The Impact of New Standards,
INFODB, Vol. 7, Part 1, pp21-30

Wight, O., 1984,
Manufacturing Resource Planning : MRP II,
Essex Junction, Oliver Wight Publications Ltd

Wilkinson, G., G., Winterflood, A. R., 1987,
Fundamentals of Information Technology
John Wiley and Sons, pp207-219

Williams, J., Rogers, P., 1991,
Manufacturing cells : control, programming and integration
Butterworth-Heinemann

Wood, P. J., Johnson, P. N., 1989,
A review of the use of SSADM and IDEF at the University of Warwick,
Procs. of SAMT' 89 Conference, Sunderland, UK

Wyatt, T., Al-Maliki, I., 1990,
Methods in manufacturing systems engineering - the background,
Integrated Manufacturing Systems, Vol. 1, No. 2, pp91-93

Yeomans, R. W., Choudry, A., 1986,
Design Rules for CIM,
North Holland

Zhang, H. -C., Alting, L.,
An Exploration of Simultaneous Engineering for Manufacturing Enterprises,
International Journal of Advanced Manufacturing Technology, Vol. 7, No. 2, pp101-108

Zäpfel, G., Missbauer, H., 1993,
New Concepts for production planning and control,
European Journal of Operational Research, Vol. 67, pp297-320

APPENDICES

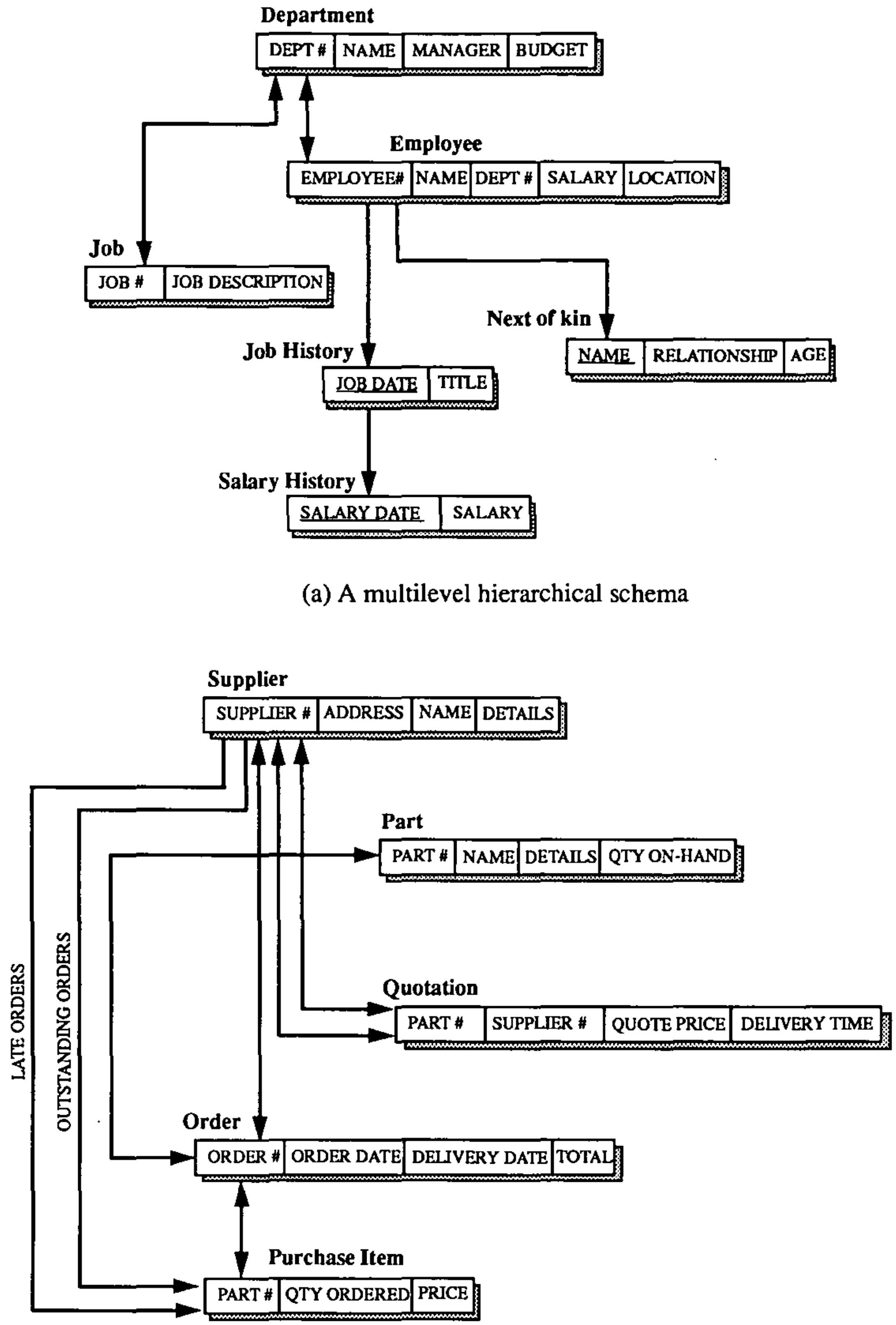
APPENDIX I ✓

Types of Logical Data Models

There are the following three logical data models most commonly supported by database management systems [Wilkinson and Winterflood 1987, Martin 1980] :

- **Hierarchical model**
  Data is represented in a hierarchical or tree structure. The highest level in the hierarchy is known as the root node. It has no parent node above it. Apart from the root node all other nodes must have only one parent node, but any node can have more than one dependent or child node. As illustrated in Figure a, node **Department** is the root, node **Employee** is the child of node **Department** and the parent of node **Job History**. Tree structures are a natural way to model truly hierarchical relationships from the real world when one-to-many (parent to child) segment types can be defined to represent successive levels in a tree structure in order to relate entities to one another. However, in many situations relationships do not naturally fit into this model. For instance it is not easy to directly represent relationships between segment types at the same level in the hierarchy, nor is it possible without introducing data duplication to represent many-to-many relationships between entities. A more detailed discussion of the hierarchical model can be found in Date [1986].

- **Network model**
  In the network model, data is represented in a network or plex structure. In the network model any node can be connected to any other node in the structure. The nodes consist of groups of data usually representing an entity and its attributes, whilst the connection between the nodes represents the existence of relationship between the nodes (entities), as illustrated in Figure b. Network structures offer more scope to represent data relationships than hierarchical structures, albeit at the expense of simplicity, at least with respect to physical storage structure. The need to transform many-to-many relationships by the construction of a network model does mean than more or less irreversible decisions have to be made about the nature of the relationships between entities when the data model is designed. It should be noted that the network model, whilst permitting a representation of many-to-many relationships without introducing duplication of the duplicating record occurrences, does make retrieval of data a laborious process. For further reading on network database systems the reader is referred to Taylor and Frank [1976], Olle [1978] and Date[1986].


Department
DEPT # | NAME | MANAGER | BUDGET
Employee
EMPLOYEE# | NAME | DEPT # | SALARY | LOCATION
Job
JOB # | JOB DESCRIPTION
Next of kin
NAME | RELATIONSHIP | AGE
Job History
JOB DATE | TITLE
Salary History
SALARY DATE | SALARY
Supplier
SUPPLIER # | ADDRESS | NAME | DETAILS
Part
PART # | NAME | DETAILS | QTY ON-HAND
Quotation
PART # | SUPPLIER # | QUOTE PRICE | DELIVERY TIME
LATE ORDERS
OUTSTANDING ORDERS
Order
ORDER # | ORDER DATE | DELIVERY DATE | TOTAL
Purchase Item
PART # | QTY ORDERED | PRICE

(a) A multilevel hierarchical schema

(b) A plex structure of five record types used for a purchasing application

Figure : Database models and structures

- **Relational model**

  In a relational model, entities, relationships and attributes are represented in the form of two-dimensional tables known as relations. Records are assimilated to the rows of the table and each set of attributes forms a column. Each row in a table is known as a tuple and consists of a fixed number of attributes. In a relational database entities are stored totally independently. That is to say the existence of a relation or a tuple in a relation is not dependent on any other relation or tuple, nor is access to a tuple reliant on explicitly pre-defined access paths through complex data structures as it is in the formatted hierarchical or network models. Instead logical associations among the stored data are exploited through relational operations, such as *select, project and join* which can be used to create new tables. The application of any (relational) operation produces an object which is itself a relation (which can be stored as a new table in the database). Thus any number of operators and relations can be combined in a 'relational expression' used to answer almost any query. The entities, attributes and relationships produced from the conceptual data model can often be modelled directly as relations in a relational database model. The use of the relational model rather than hierarchical or network models is seen to demand less compromise in transforming the real-world model of the conceptual data model, although the processing overhead it requires is still often a serious deterrent to its use for many applications [Date 1986].

APPENDIX II ✓

The following publications relate to the CIM Model Project :

"CIM Model For Metal Machining Trade - Translating Vision into Reality"
Valdew Singh
Proceedings of International Conference on CIM (ICCIM' 91),
October 1991, Singapore, pp336-341

"Flexible Materials Handling and Storage System for Integrated Manufacture"
Valdew Singh
Proceedings of Asia Pacific Industrial Automation (IA' 92) International Conference,
May 1992, Singapore, pp10-21

Proceedings of International Conference on CIM (ICCIM' 91),
Oct 1991, Singapore

CIM Model for Metal Machining Trade - Translating Vision into Reality

Valdew Singh
Applied Technology Group
Singapore Economic Development Board

ABSTRACT

This paper describes the effort involved in developing a Computer Integrated Manufacturing (CIM) model which will serve as a working showcase to the local industry. It comprises a defined hierarchy of control and integrated information flow which links very closely the various production planning and shop floor related activities, and a flexible manufacturing cell serviced by an automated material handling system that includes an AGV and ASRS. A detailed overview of the realised system and the functional aspects and requirements of the CIM modules will be provided.

INTRODUCTION

An immediate problem facing Singapore is the shortage of labor supply and according to the National Automation Master Plan (1988) Report [1], automation is the key technology to improving labor productivity, flexibility, enhancing competitiveness, and stimulating growth in the future. CIM technology, therefore, has definitely a very important role to play towards ensuring the continued survival and helping to sharpen the competitive edge of manufacturing companies.

However, its vision seldom translates into reality primarily due to the lack of awareness, proper training as well as standards and open structure in the use of multi-vendor systems to integrate and interface the various "islands of automation".

In order to help promote greater awareness and better expose industry, particularly in the metal machining trade, to CIM technology, a project to develop a CIM model [2] was initiated by the Applied Technology Group (ATG) of EDB; vehicle for the promotion and propagation of new application technologies.

The emphasis for the CIM model is on the development of a strategy to guarantee a constant, efficient and distributed information flow and processing to link all production related activities. It also comprises a flexible manufacturing cell (FMC) responsible for the machining activities and an automated material handling system that includes an automated guided vehicle (AGV) and an automated storage and retrieval system (ASRS) to offer transportation and storage support to the FMC.

APPROACH FOR IMPLEMENTATION

As a guideline, the following is the approach taken to initiate the implementation of the CIM model :

- Rationalization to identify and understand the specific needs and relate them to already available technology for selection and adoption.
- The analysis and simplification of all existing business and manufacturing activities to eliminate counter-productive weak points such as double tracking and dead ends to achieve efficiency through optimization of these activities.
- Automation and computerisation is then initiated. Full compatibility and ease of customisation must always be considered for future integration of these newly created "islands of automation". The initial approach to computerisation is via the MRP II system to allow automation of labor intensive and time consuming processes, purchase order processing, routing, bills of material pre calculation and MRP.

- The integration of the "islands of automation" is carried out in progressive stages to make it manageable and to help isolate and tackle any problem that might arise.

MODULES INCORPORATED FOR THE CIM MODEL

The configuration of the CIM model is illustrated in Fig. 1 and it includes the following modules :

In-house developed modules

Shop floor Scheduler	Finite capacity planning and loading.
Tool, Fixturing & Material Management (TFM)	Check on the machining parameters and the availability and inventory status of selected tools, fixtures and materials. The TFM system is linked directly to MRP II.
Product Assessment	To selectively identify and accept suitable product for processing.
Remote DNC	Network based Distributed Numerical Control (DNC) system linked to the DICON DNC manager for NC program management and automatic control and monitoring of the CNC and inspection stations.

The lack of suitable commercial systems which could offer the required openness and functionality that is encompassing enough for the needs of the user has made it necessary for these in-house developments.

Off-the-shelf commercially available modules

MRP II system *** Fourth Shift**	Production management, planning and control.
CAD *** AUTOCAD**	Generation of engineering drawings.
CAM *** EZ-CAM**	Generation of NC programs.
CAD/CAM Management *** COMPASS**	To archive and manage engineering drawings and NC programs.
Real-time Shop floor Monitoring & Control *** ONSPEC**	Provide status feedback & event triggering to initiate shop floor activities in a coordinated manner. It is carried out through direct communication with the SIEMENS PLC which is networked with the SINEC L1 Local Area Network to offer distributed PLC control.
Simulation *** GRASP** *** SIMFACTORY**	Graphical and discrete event simulation to model and analyze the operational strategies of the shop floor manufacturing activities.
Quality Control *** CVQA**	Generate inspection program for CMM

These modules, however, exist as stand-alone "islands of automation" and they need to be specially customized and bespoke software and macros have to be provided to offer the opportunity for these modules to be integrated and interfaced.

All the CIM modules are linked via the NOVELL Local Area Network (LAN) for information flow, refer to Fig. 1 for illustration. The following LAN solution has been adopted to allow intercommunication between the various in-house developed and heterogeneousmix of selected multi-vendor systems which operate over varied platforms which includes MD-DOS, OS/2 and UNIX :

- Netware protocol between all MS-DOS and OS/2 based systems. In addition, the MAP ASSIST utility has been utilized to offer peer to peer communication over NOVELL LAN between the MS-DOS based systems. This effectively help reduces the dependency on the network server and the information traffic flow to the server.
- 3+Open TCP with DPA (Demand Protocol Architecture) for communication between the MS-DOS and OS/2 based systems with UNIX.

Based on this LAN solution, a database strategy is developed to support a distributed information processing architecture. This is to facilitate the integration of all dissimilar, non-standard and local databases which are proprietary to the respective application software in use to a more standard and shareable DBMS (Database Management System) [3,4,5]. ORACLE RDBMS has been chosen to serve as the required DBMS because of its portability over MS-DOS, OS/2 and UNIX platforms, data and security control across distributed platforms and flexibility as well as ease in application development and structuring because of SQL support. User friendly front end interfaces have been developed through ORACLE to allow easier and common access and storage of information by the various application software. Tracking, confirmation and logging in of all the planned activities and relevant information will be undertaken by it.

MANUFACTURING CELL

The layout for the FMC is illustrated in Fig. 2. The cell includes two CNC machining centres (i.e. MAHO MC5 and MHO MH700S), and automated material handling system which includes an AGV and an ASRS. The machines are linked to the remote DNC terminals which serve as cell controllers. It was necessary to enhance and retrofit these stand-alone CNC machines for unmanned machining operation through the provision of the following :

- Modular docking cum buffer stations for the receipt and delivery of pallets from and to the AGV.
- Automated transfer mechanism for the transfer of the pallets between the docking stations and the machines.
- Automated clamping and location for holding the pallets securely when machining.

All these automated systems are under the control of the networked based SIEMENS PLC.

The FMC is able to function in an autonomous manner without much operator intervention because of the following :

- Ease of handling a defined family of products with minimum setup and transfer time due to the multi-axis machining capability.
- The communication protocol available on the CNC controller allows micro-computer control for :
  - Automatic start-up, NC program selection, upload and download.
  - Vertical integration of the cell with the real-time shop floor monitoring and control software and the remote DNC system for status feedback and automated control and coordination of the cell.

The parts will be fed into the ASRS for storage via a manual fixturing station where they will be fixtured with the aid of modular fixtures on common size pallets [6]. The ASRS will be responsible for the storage of empty pallets, prefixtured parts that are ready for machining operations but need to be stored temporarily prior to dispatch. The AGV is equipped with a twin conveyor carrier to handle two pallets simultaneously. It will automatically shuttle pallets between the ASRS and the FMC

The Carl Zeiss UMC 550S coordinate measuring machine (CMM) is responsible for inspection of the parts that has been machined. The parts to be inspected will be delivered by the AGV from the ASRS to the specially developed buffer cum transfer mechanism for automated transfer to the CMM.

INFORMATION FLOW

The CIM environment created represents what happens when an order is placed and the logic that follows. With reference to Fig. 3, the information generated and the sequence of flow between the modules are as follows :

• Product Assessment - Check for suitability

The order received is interactively assessed for suitability. Existing constraintsand specifications must be satisfied such as the size of part, payload, batch quantity and type of material permitted and the machining processes that can be accommodated by the system. Suitable order will be further processed.

• MRP II - Order Entry

The order is registered and availability check on the required machining capacity and other resources necessary to manufacture the order will be performed. The route plan will be generated and a delivery date is provided after considering the manufacturing and delivery lead times for the required items as well as the loading commitment for all prior confirmed orders.

The quotation will also be generated based upon the available costing information. If the quote and delivery date is agreeable to the customer, the process of product design will commence.

• CAD - Product Design & Engineering Drawing Generation

Within the CAD/CAM manager environment, all part drawings will be generated. The TFM software has been directly interfaced to provide on-line information on available toolings, fixtures and materials for possible selection and consideration.

The CAD system has been customised to include a standard library for the modular fixturing elements so as to allow computer aided fixturing to be carried out interactively to generate the necessary fixturing drawing for the part concerned.

The part drawing has to be converted to either the neutral Data Exchange Format (DXF) or Intermediate Graphic Exchange Standard Format (IGES) for input to the EZ-CAM and the CVQA inspection software for subsequent NC and inspection program generation respectively.

The generated part and fixturing drawings as well as the DXF/IGES converted file will be stored in the CAD/CAM manager database located on the network server.

The BOM (Bill of Materials), specifying all materials and standard items required for the part as well as the necessary modular fixture elements, will be created through the BOM module of the MRP II system residing within the CAD/CAM manager.

• CAM - NC & Inspection Program Generation

EZ-CAM also operates within the CAd/CAM manager environment. It imports the necessary IGES or DXF formatted part drawings to post-process the required NC program. Similarly, the CVQA software will post-process the IGES formatted part drawings to generate the inspection program necessary for the CMM. The NC and inspection programs will be sent to the DICON DNC manager for storage until required.

The BOM, specifying all tooling requirements, will be created through the BOM module of the MRP II system residing within the CAD/CAM manager.

• MRP II - Generation of Manufacturing & Purchase Order

The MRP II system will automatically generate action messages for the purchase of relevant items after receipt of the full BOM for the part concerned. The order will be confirmed and incorporated into the Master Production Schedule (MPS).

The manufacturing order and a picklist, which provides information on all the resources needed to produce the part, i.e. materials, toolings, fixtures, etc, will be generated. The manufacturing order for the part will be input automatically to the shop floor scheduler for further processing.

The picklists and the manufacturing orders will only be released when the availability of materials, toolings, fixtures, process plans, Nc programs, and part and fixturing drawings are confirmed.

• Shop floor Scheduler - Generation of Scheduled Orders

The shop floor scheduler will perform finite scheduling of the manufacturing orders received from the MRP II system. It is heuristic based [7, 8] and considers criteria such as critical ratio and slack time in obtaining a feasible solution fast. It will generate the following three schedules for :

- Processing of parts stored in the ASRS.
- Prefixturing of parts for forward loading.
- Processing of the batch orders for each of the specific stations.

The shop floor scheduler is linked to the simulation software, SIMFACTORY, in order to determine optimum capacity loading and resource allocation. This is to alleviate potential problems such as bottlenecks as well as poor and unbalanced utilization of machines. The scheduled orders will be held in queue until they are to be released depending upon the start date.

The scheduled lists for the processing of the parts stored in the ASRS and prefixturing of parts for forward loading will be accessed by the ASRS controller for processing based upon the availability of the stations. The lists will be updated by the ASRS controller in order to allow order monitoring for the following :

- Comparison of actual to planned quantity status.
- Indication of the operational status for the order.

• Remote DNC - NC Program Management & Cell Control

The remote DNC terminal attached to the specific machine will access the scheduled list containing the batch of orders to be processed by it. It will upload the associated NC programs from the DICON DNC manager. The terminal will coordinate the sequence of activities necessary for unmanned machining operationsuch as sensing for pallet presence and checking to ensure that the pallet has been properly secured prior to activating the machining cycle. An optical based part identification (ID) system, SUNX/IDX, is interfaced to the terminal to allow the reading of the ID tag, which is attached to the pallet. This is to check the authenticity of the part received and relate it to its NC program for machining. The part ID system allows traceability of the part as it flows through its planned route.

• MRP II - Close Order

Information concerning the completion of an order will be captured by the shop floor scheduler from the updated schedule lists. It will automatically update and inform the MRP II system. The MRP II system will then generate action messages which can be acted upon to close the order. The completed order is then ready for dispatch to the customer.

CONTROL ARCHITECTURE

The control architecture has been formulated to cater for the processing of the scheduled orders based on the pull-through technique; i.e. required orders and services will only be provided when the need arises. This is to allow balanced and optimum utilization of resources.

The ASRS controller will process the orders from the scheduled lists depending upon the availability of the buffering cum docling stations at each of the machining and inspection stations. This information is feedback on-line by the real-time shop floor monitoring and control software, ONSPEC. After dispatch of the order the ASRS will register a request for the service of the AGV to collect and deliver the part to its designated station. The AGV will be informed by ONSPEC to collect parts that have been machined and inspected.

This network based heterogeneous control architecture [9] has been adopted over the more pervasive hierarchical architecture so as to cater for effective distributed information processing. Each entity will be able to operate independently with intercommunication on a peer to peer basis. Furthermore, such an architecture allows for ease of software development, maintenance, system reconfiguration and expansion.

CONCLUSION

In summary, it is hoped that this model can help serve as means to an end in keeping pace with the constantly evolving CIM technology and to offer assurance and confidence to those wishing to venture into CIM.

The future plan is to incorporate more intelligence into the system to further isolate and contain menacing production variables and to enable it to operate with greater autonomy in the acquisition and processing of the necessary information across multi-vendor platforms.

ACKOWLEDGMENT

The author wishes to express his thanks to the Manpower Development Division of the Singapore Economic Development Board for much appreciated support. Special thanks are also expressed to his colleagues in ATG involved in the project (in particular C. N. Chang, S. D. Foo, N. F. Choong, S. H. Ang, K. S. Tam, C. K. Lim and C. M. Ching) and to his family, Magdalene and Sabrina, for their understanding, patience and constant encouragement.

REFERENCES

1. Productivity Digest/ SME Newsletter, March 1989

2. Singh, V., "PC Based Computer Integrated Manufacturing Solution for Small and Medium Enterprise", Proceedings of the Asia-Pacific Industrial Automation (IA' 90) Conference, May 1990, Singapore

3. Ranky, P. G., "Manufacturing Database Management and Knowledge Based Systems" CIMware LTD, Guildford, Surrey (UK), 1990

4. Yourdon, E., "Modern Structure Analysis", Prentice-Hall, Eaglewood Cliffs, New Jersey, 1989

5. CIM Strategies, "Client-Server Architecture, What does it buy you?" Vol. VIII, No. 3, March 1990

6. Lewald, R., "Soon : an international pallet standard", American Machinist, Feb 1990, pp69-70

7. Berry, W. L., Rao, V., "Critical ratio scheduling", Management Science, Vol 28, No. 2, 1975

8. Chang, Y. L., Sullivan, R., "Schedule generation in a dynamic job shop", International Journal of Production Research, Vol 28, No. 1, 1990

9. Duffie, N., A., Piper, R. S., "Non-Hierarchical Control Model of a Flexible Manufacturing Cell", Robotics & Computer Integrated Manufacturing, Vol. 3, No. 2, pp175-179

REGISTERED TRADEMARKS

SIMFACTORY - CACI SimLab, Inc

GRASP - BYG Systems Ltd, U.K.

ORACLE - ORACLE Corp

ONSPEC - Heuristics, Inc

NOVELL - Novell, Inc

FOURTH SHIFT - Fourth Shift Corp.

3+Open - 3COM Corp

MAP ASSIST - Fresh Technology Corp

AUTOCAD - Autodesk , Inc

EZ-CAM - Bridgeport Machines, Inc

COMPASS - TCAE GmbH

CVQA - Computer Vision, Inc

DICON - Dinkel Industrie Automation GmbH

Proceedings of International Conference on CIM (ICCIM' 91),
Oct 1991, Singapore

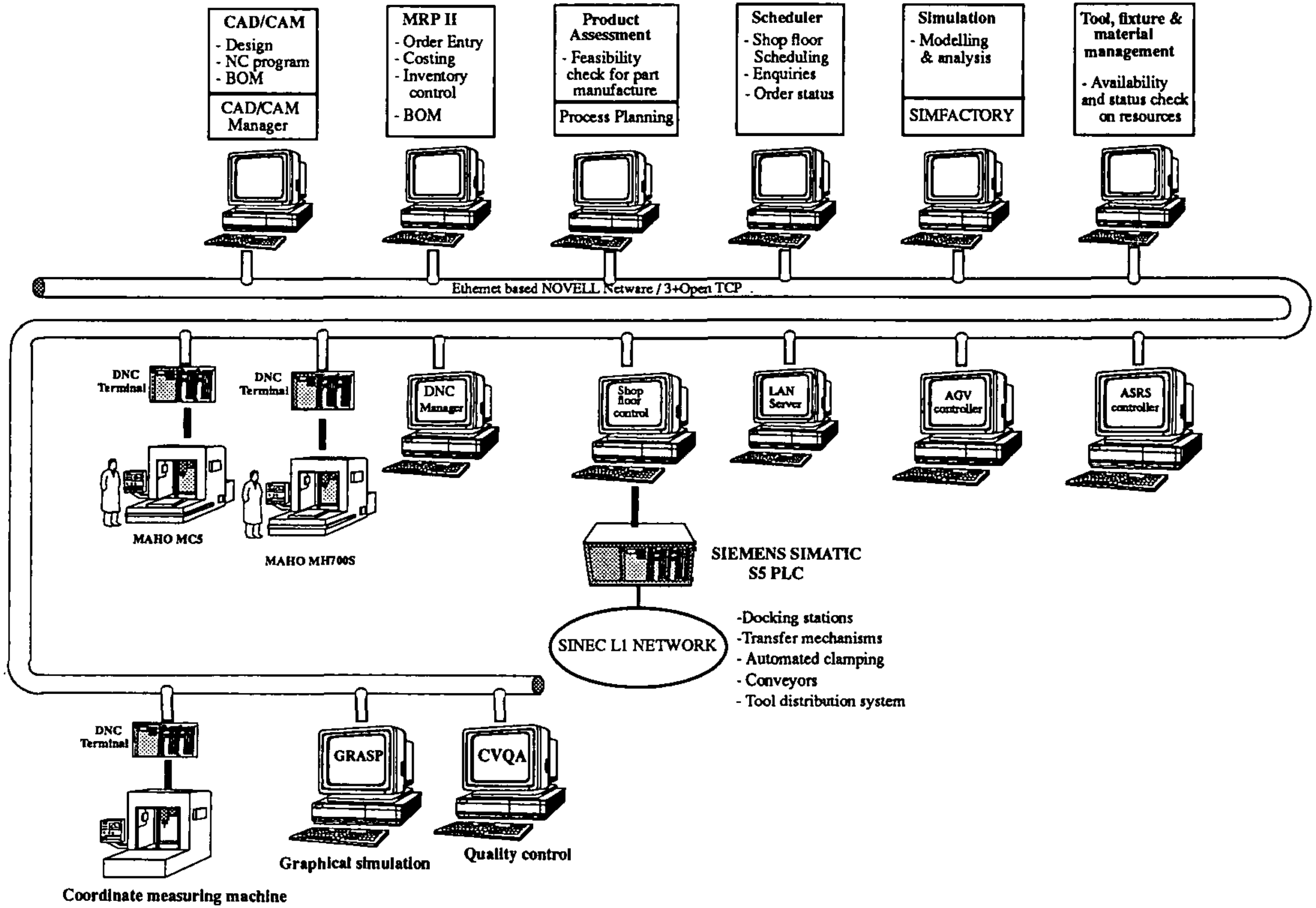


Fig 1 - CIM MODEL CONFIGURATION

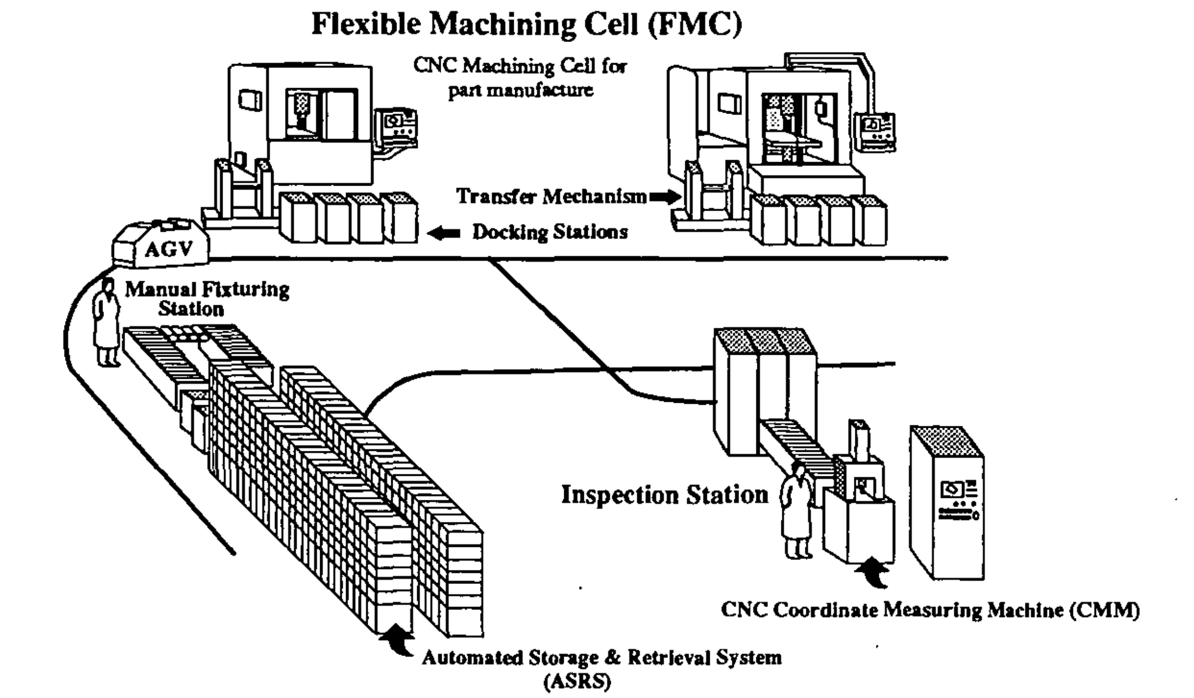

Flexible Machining Cell (FMC)
CNC Machining Cell for part manufacture
Transfer Mechanism
Docking Stations
AGV
Manual Fixturing Station
Inspection Station
CNC Coordinate Measuring Machine (CMM)
Automated Storage & Retrieval System (ASRS)

Fig. 2 - Physical Layout of Flexible Manufacturing cell (FMC)

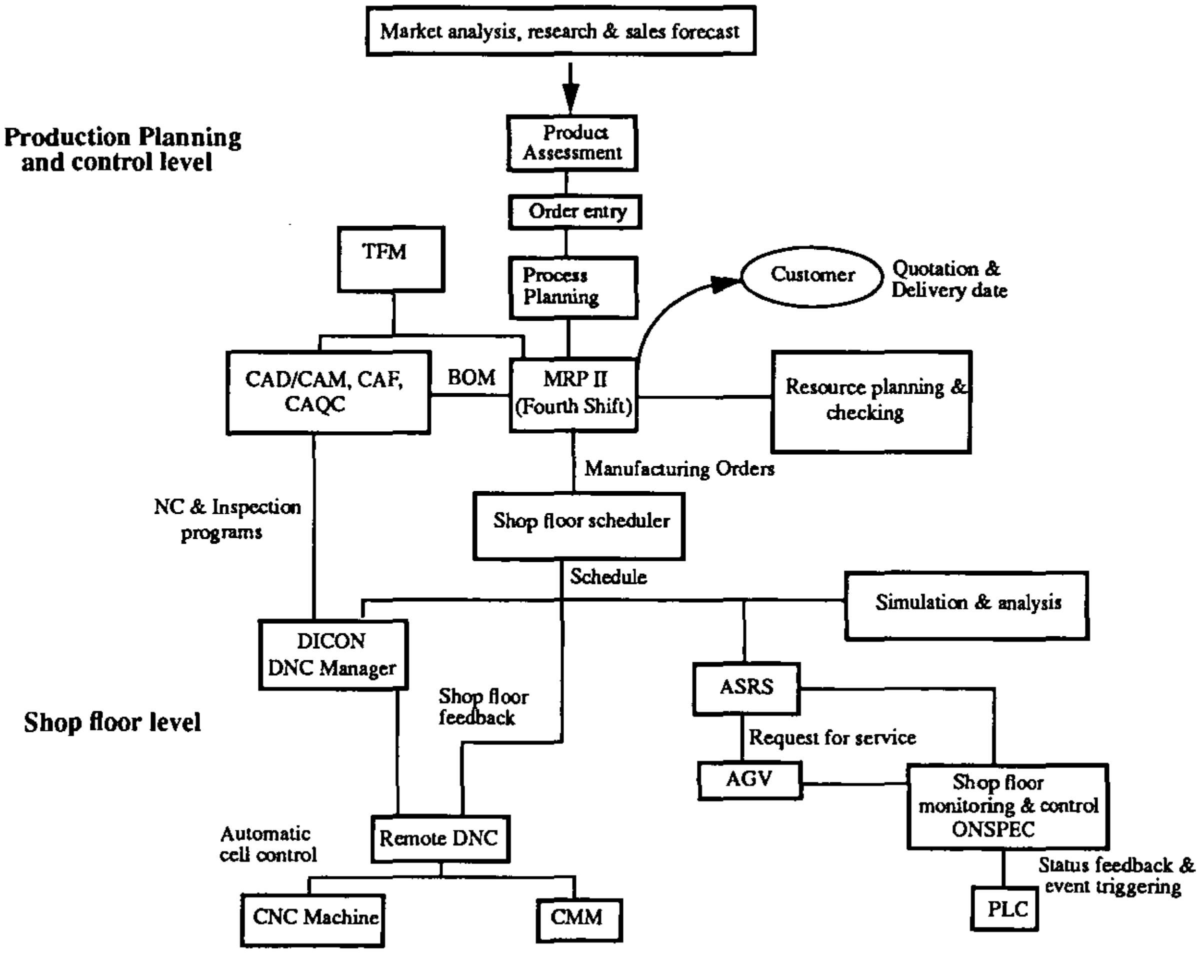


Fig. 3 - Information flow

Proceedings of Asia Pacific Industrial Automation (IA' 92) International Conference,
May 1992, Singapore

Flexible Materials Handling and Storage system for Integrated Manufacture

Valdew Singh
Applied Technology Group
Singapore Economic Development Board

ABSTRACT

The flow and management of material and information in material handling system are proving to be technologies with the most dramatic impact on integrated factory automation. In estimated between 70 and 80 % of throughput time during manufacturing can be accounted for by processes such as transport, handling, buffering and storage.

A flexible material handling and storage system (FMHSS) has been developed as part of a project to develop a computer integrated manufacturing (CIM) model which will serve as a working showcase to the local industry. The FMHSS comprises an AS/RS (Automated Storage and Retrieval System) and two AGVs (Automated Guided Vehicle) to provide storage and transportation services respectively to a flexible machining cell (FMC), flexible assembly cell (FAC) and an automated CMM inspection station. The objective of the FMHSS is to supply the right quantity of parts in an efficient, controlled and coordinated manner at the right time. The FMC has been retrofitted with specially designed clamping and transfer system for unmanned precision machining and automated part transfer. The FAC comprises two robotic system working in close coordination for automated assembly of a family of components. The AS/RS cater for parts and component for both FMC and FAC. The AGVs offers controllability and fast reaction to flexibly interlink the AS/RS, FMC, FAC and the CMM inspection station. The FMHSS is driven by schedules and shop floor status feedback generated by an integrated production planning and management system.

This paper describes the effort involved in the development of the FMHSS and it will highlight the following :

- The modular approach for progressive implementation of the various individual material handling modules with the aim of combining them into an integrated system.
- The network based control architecture which involves an ethernet based local area networking for distributed information processing and PLC networking for shop floor automation.
- The adoption of the pull-through technique where the FMC and the FAC will be services by the AS/RS and the AGVs based upon their availability for part processing.
- The integration of the AS/RS warehouse management software with the production planning and management systems such as MRP II and the Shop floor Scheduling through a SQL based relational database management system (RDBMS).

1.0 INTRODUCTION

In the local manufacturing industry today, the investment in factory automation such as the application of automated machine tools and computer-aided applications, e.g. CAD/CAM systems, to improve productivity is pervasive because of the following reasons :

- Growing problem of shortage and high cost of skilled labour.
- Intense competition among companies to gain the competitive edge.
- Maturing of related technologies, such as computer numerical control (CNC), programmable logic controllers (PLC), computer and electronic communication technology.

However, the level of automation among the local small and medium enterprises (SMEs) [1] is generally confined to stand-alone automatic machines and computer-aided applications where implementation is usually not well planned and is often carried out on a piece meal basis. This can only offer short term benefits with limited opportunity for progressive growth and enhancement.

Therefore, to help introduce and create greater awareness among the local companies to the concept of CIM (Computer Integrated Manufacturing) a CIM model [2,3] was developed by the Applied Technology Group (ATG) of EDB; vehicle for the promotion and propagation of new application technologies.

The model incorporates a flexible machining cell (FMC) for precision machining and a flexible automated cell (FAC) for automated part assembly. Its emphasis is on the strategic application of computer-aided technologies to form an information network to combine the activities relating to engineering, business and production functions required for product planning and manufacture. The distribution of activities within the hierarchy of this CIM model is illustrated in ANNEX I.

As the prime task for the realisation of this model lies in the integration of both information and material flow, the inclusion of Automated Guided Vehicle Systems (AGV) and Automated Storage and Retrieval Systems (AS/RS) are necessary as these are fast becoming indispensable components in integrated factory management and automation. These technologies facilitate automated movement of products and materials according to planned schedule and provide effective management and control to help optimise production.

2.0 FLEXIBLE MATERIALS HANDLING & STORAGE SYSTEM

As illustrated in ANNEX II, the following are the major components of the flexible materials handling and storage system (FMHSS) for the developed CIM model :

- **Flexible Machining Cell (FMC)**

The FMC includes the MAHO MC5 and MH700S CNC machining centres for the fabrication of a defined family of products. These products will be mounted on common sized pallets [4] with the aid of modular fixtures. The Carl Zeiss UMC 550S coordinate measuring machine (CMM) is incorporated for quality control and automated part inspection. These stand alone machines are dissimilar are linked to remote Direct Numerical Control (DNC) terminals which serve as cell controllers. In order to enable the cell to function in an autonomous manner without much operator intervention the following enhancements and provisions were necessary :

- Buffering cum AGV docking stations to service CNC machines
These will serve as intermediate buffer areas for the receipt and delivery of pallets from and to the AGV. They are made of standard height for the benefit of the AGV and are modular in design.

-Automated pallet transfer units
It is responsible for the transfer of pallets between the AGV docking station and the CNC machine. It is made mobile over a short linear track and is controlled by the linear axis motion control module of the PLC. This makes the location programmable and reconfigurable if necessary. It also provides the necessary height adjustment between the AGV docking and the height of the CNC machine table during the transfer of pallets.

- Swarf control & automated pallet clamping and location

- CMM conveyor system
A conveyor system for pallet transfer has been developed and incorporated with the CMM. It allows the automated transfer of pallets from the AGV to the CMM and vice-versa.

- Upgrading of CNC machine controllers
The MAHO machine controllers were upgraded to allow for the provision of the LSV2 communication protocol. Through the remote DNC terminals specific communication routines have been written based on the LSV2 protocol to address the required machine functions in order to allow for remote control of the CNC machines such as automatic start up, upload, download and selection of NC programs.

- PLC for FMC hardware automation

The SIEMENS SIMATIC S5 PLC 115U is used for sequence control of all sensors, actuators, limit switches and other devices that would require digital I/O control for the FMC. The SIEMENS SIMATIC S5 L1 PLC bus network is used to link the various automated modules in the FMC. The master PLC in the network is linked to the microcomputer based shop floor control and process monitoring software, ONSPEC, for status feedback and event triggering for the FMC. Refer to ANNEX III for illustration.

- Part Identification

Part identification (ID) within the FMC is provide by the optical based SUN X/ID system which allows for read/ write of information such as routing and operation sequences to special ID tags which are embedded and carefully concealed at the side of the pallets. The ID read/write scanners are locate at the various processing stations to check for part authenticity during receipt of the pallets as well as to update the operation status of the part after it has completed each operation. It allows for ease of part traceability and error recovery.

• Flexible Assembly Cell (FAC)

The FAC has been configured to include the BOSCH & SKILAM pick and place robotic systems to perform assembly activities for the family of components to be stored on pallets in magazine. The FAC is capable of the following :

* Ease of handling a defined family of products with minimum setup and transfer time without operator intervention.
* Flexibility in programming and movement of the robot to avoid restriction for component handling and orientation.
* Integration of the two robotic systems with the automated conveyor system for transfer of magazine within the FAC via the PLC.
* Ability to control and monitor the activities and status of the robotic cell through the in-house developed EYESCREAM based shop floor control and process monitoring application. This offers the opportunity for vertical integration for status feedback and information processing.
* Magazine transfer through the provision of automated magazine transfer and buffering facilities for unmanned assembly for receipt and delivery of magazine between the assembly stations and the AGV.

The OMRON PLC is used to link the various automated modules for the FAC. The OMRON SYSLINK RS422 link is used to coordinate and synchronize the activities of the BOSCH and the SKILAM robotic systems with the automated conveyor system for transportation of pallets and magazine for automated part assembly. It is linked to EYESCREAM based shop floor control and process monitoring software.

• AS/RS and AGVs

The AS/RS is for common storage of pallets and magazine to service both the FMC and FAC. It consists of two racks facing each other with the stacker crane servicing the 216 available storage compartments. Eighty of the storage compartments are reserved exclusively for the storage of magazine for the FAC. There are two AGVs. One AGV is equipped with a twin conveyor carrier dedicated to shuttle pallets between the AS/RS and the FMC and the other has a two tier conveyor system to transport magazine between the FAC and the AS/RS. The AS/RS has been retrofitted with the following in-house developed transfer system to support its automated material handling functions:

* Modular chain driven conveyor system equipped with input and output bays for store-in and store-out of pallets for FMC.
* A two tier roller conveyor system for store-in and store-out of magazine for FAC.

Since the AS/RS has to service the FMC and FAC simultaneously, its control and inventory management application software has, therefore, been developed under the UNIX platform to take advantage of the necessary multi-tasking and multi-user functions. The ORACLE relational database management system has been chosen as the database for the AS/RS system because of the following:

* Openness for access of relevant data for processing and manipulation through SQL.
* Ease of report generation and formatting.
* Portability and ability to intercommunicate across heterogeneous platforms.

The AGVs rely on magnetic guidance to move along special magnetic based tracks laid on the floor to define the AGV routing. They are capable of bidirectional and transverse movements with spin turn ability. The AGVs are interfaced with all the docking stations through a special 4-bit optical based sensor. This sensor serve as an intermediate communication interface between the AGV and the PLC module controlling the docking station. It helps to synchronize pallet and magazine transfer between the AGV and the docking stations.

3.0 FMHSS PLANNING AND DESIGN

3.1 Graphical Simulation

The GRASP computer-aided graphical simulation package has been utilized to build a model of the FAC, FMC, AS/RS as well as the two AGVs for visualization and dynamic simulation of the material flow. It serves as a useful tool to help analyse the design of the various automated material handling modules and enables the simulation of the operational performance. The derivatives from this graphical simulation exercise includes the following :

- Optimum layout planning.
- Collision detection of all interacting facilities.
- Determination of parameters such as operation cycle, transfer and idle times.
- Definition for operation sequence and digital I/O specification for overall PLC program consideration.

3.2 Part Family specification

Based upon the physical specifications as derived from the FMC and FAC and historical production records; statistics were obtained to determine the characteristics of the parts that have been produced such as maximum and minimum physical size of parts that can be accommodated, tooling and material requirements, number of settings required, processes supported, and etc. This information is used to formulate the part families and based upon these information a Product Assessment Module has been developed to assess the suitability of parts and only parts that meet the characteristics for the part family will be accepted. This is essential in order to match process and operation requirements to available capability and resources.

4.0 CONTROL ARCHITECTURE FOR FMHSS

The description for the flow of information within the control architecture of the FMHSS will commence from the receipt of the daily schedule lists by the AS/RS controller from the Scheduler for machining and assembly. These schedule lists have been optimised [5] based on criteria such machine utilisation, throughput time and work-in-progress.

4.1 Operation Procedure

The operation begins with raw materials pre-fixtured on common size pallets and components loaded into the magazine at the manual fixturing and assembly stations and stored away in the AS/RS. Parts will be retrieved from the AS/RS and sent to the respective machining centre, assembly station and the CMM for processing, and finally sent back to the AS/RS for either in progress or temporary storage. Parts that require secondary operations like grinding or heat treatment, are sent out of the system for these operations to be accomplished and returned finally to the AS/RS.

4.2 Information Processing

The AS/RS controller needs to shuttle between servicing the FAC and FMC. It will only store-out the orders from the scheduled lists depending upon the availability of the machining, assembly and inspection stations. This information is feedback on-line by the real-time shop floor control and process monitoring software, ONSPEC, which is responsible for the initiation, coordination and synchronization of all FMHSS activities. After dispatch of the order the AS/RS will register a request for the service of the AGV to collect and deliver the parts to its designated station. The AGV will be informed by ONSPEC to collect parts that have been machined, assembled, and inspected. Refer to ANNEX IV for illustration of the information flow for the FMHSS.

The AS/RS and AGV controllers, ONSPEC and the shop floor control and process monitoring application for FAC are integrally linked over a NOVELL ethernet based local area network (LAN) backbone. The following were incorporated as part of the LAN solution in order to allow for intercommunication and information transfer between the specific applications across heterogeneous platforms :

- Netware protocol between all the MSDOS and OS/2 based systems. In addition, the MAP ASSIST utility has been used to offer peer to peer communication over NOVELL LAN between MSDOS based systems. This effectively reduces the dependency on the LAN server and the information traffic flow to the server.
- 3+Open TCP with DPA (Demand Protocol Architecture) for communication between the MSDOS and OS/2 based systems with UNIX.

Based upon this LAN solution the control architecture has been formulated to cater for the processing of the scheduled orders based on the "pull-through technique" where parts will be fabricated and assembled only upon availability of the processing station. This is to allow balanced and optimum utilization of resources.

This heterogeneous control architecture has been adopted because it caters for effective distributed information processing. Each entity will be able to operate independently with intercommunication on a peer to peer basis. Furthermore, such an architecture allows for ease of software development, maintenance, system reconfiguration and expansion.

5.0 CONCLUSION

In summary, the developed integrated material handling system offers flexibility in terms of the following :

- **Modularity**
- **Programmability**
- **Reconfigurability**

This is essential in order to allow for progressive implementation, ease of adaptation to suit changing product and process needs, and more importantly to safeguard the system against technological redundancy and obsolescence. This system will serve as a working technological showcase to help promote the concept of integrated manufacturing to the local manufacturing industry.

6.0 ACKNOWLEDGMENT

The author wishes to express his thanks to the following :

- The Manpower Development Division of the Singapore Economic Development Board for much appreciated support.
- Colleagues in ATG involved in the project (in particular N.F. Choong, S.H. Ang, C.K. Lim and C.M. Ching)
- Paul Binding and Yamanouchi from ER Mechatronics Pte Ltd.

Special thanks are also express to his wife Magdalene for her constant encouragement, patience and kind understanding.

REFERENCES

1. Productivity Digest/SME Newsletter March 1989.

2. Valdew Singh, " *PC Based Computer Integrated Manufacturing Solution for Small and Medium Enterprise*", Proceedings of Asia-Pacific Industrial Automation (IA) '90 Conference, May 1990.

3. Valdew Singh, " *CIM Model for Metal Machining Trade*", Proceedings of International Conference on CIM (ICCIM) '91 Conference, Oct 1991.

4. Roon Lewald, "*Soon: an international pallet standard*", American Machinist, Feb 1990, pp 69-70.

5. NF Choong, KP Ang, V. Singh, " *Computer Simulation as a Tool for Integrated Manufacturing*", Proceedings of Asia-Pacific Industrial Automation (IA) '92 Conference, May 1992.

REGISTERED TRADEMARKS

SIMFACTORY - CACI SimLab, Inc

GRASP - BYG Systems Ltd, U.K.

ORACLE - ORACLE Corp

ONSPEC - Heuristics, Inc

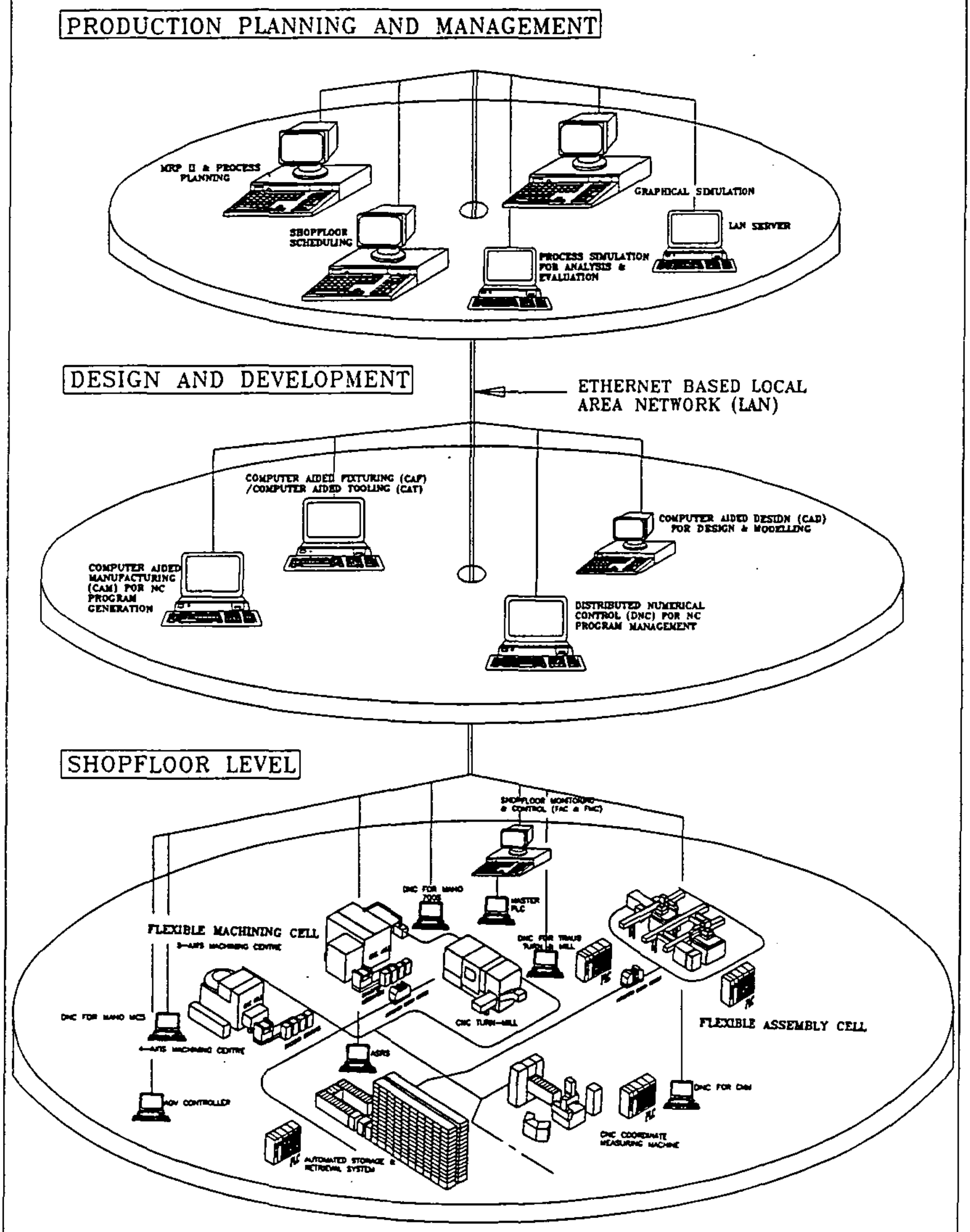
NOVELL - Novell, Inc

EYESCREAM - Real Time Graphics, Inc

3+Open - 3COM Corp

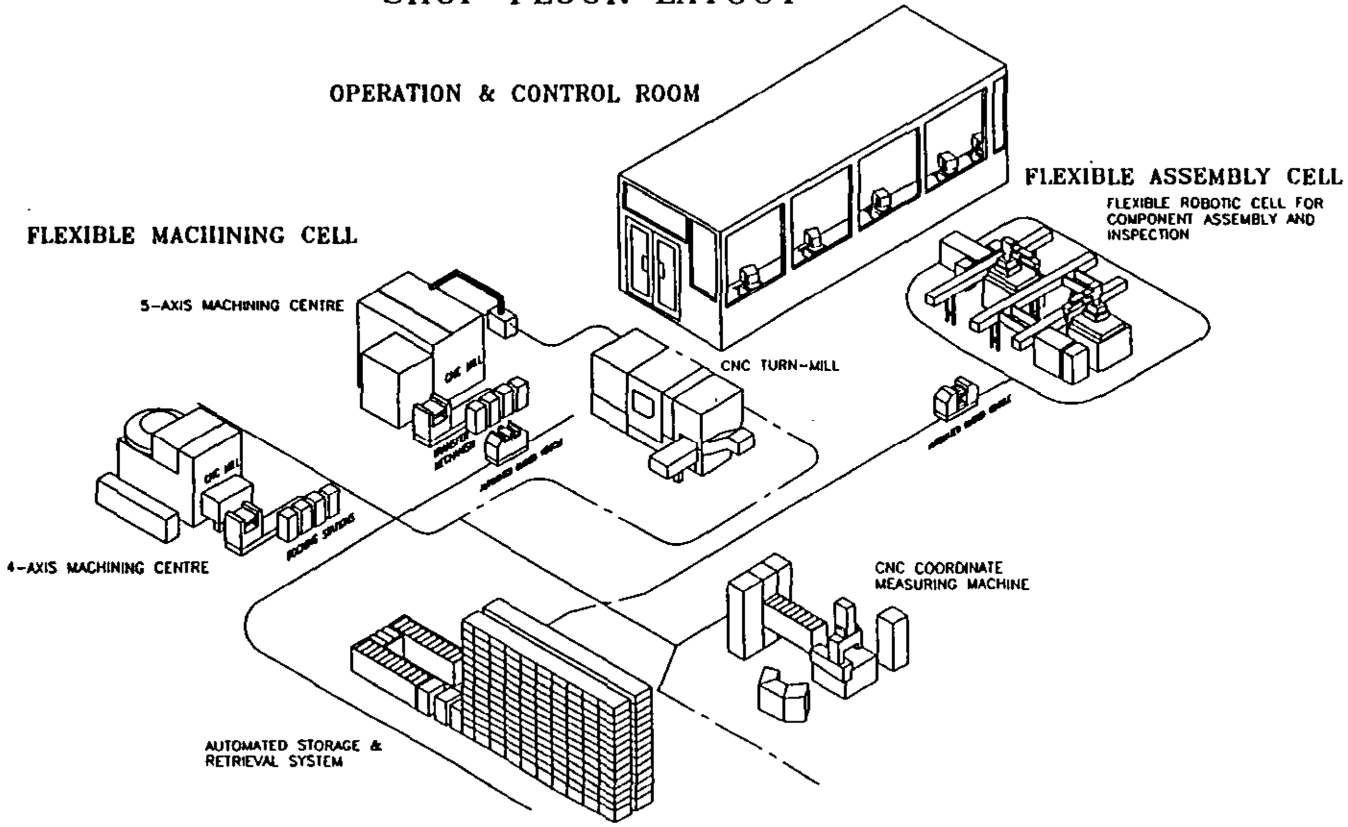
MAP ASSIST - Fresh Technology Corp

ANNEX I


PRODUCTION PLANNING AND MANAGEMENT
MRP II & PROCESS PLANNING
GRAPHICAL SIMULATION
SHOPFLOOR SCHEDULING
LAN SERVER
PROCESS SIMULATION FOR ANALYSIS & EVALUATION
DESIGN AND DEVELOPMENT
ETHERNET BASED LOCAL AREA NETWORK (LAN)
COMPUTER AIDED FIXTURING (CAF) /COMPUTER AIDED TOOLING (CAT)
COMPUTER AIDED DESIDN (CAD) FOR DESIGN & MODELLING
COMPUTER AIDED MANUFACTURING (CAM) FOR NC PROGRAM GENERATION
DISTRIBUTED NUMERICAL CONTROL (DNC) FOR NC PROGRAM MANAGEMENT
SHOPFLOOR LEVEL
SHOPFLOOR MONITORING & CONTROL (FAC & FMC)
DNC FOR MAHO 700S
MASTER PLC
FLEXIBLE MACHINING CELL
3-AXIS MACHINING CENTRE
DNC FOR TRAUB TURN & MILL
PLC
DNC FOR MAHO MCS
CNC TURN-MILL
PLC
FLEXIBLE ASSEMBLY CELL
4-AXIS MACHINING CENTRE
ASRS
DNC FOR CMM
AGV CONTROLLER
PLC
CNC COORDINATE MEASURING MACHINE
PLC
AUTOMATED STORAGE & RETRIEVAL SYSTEM

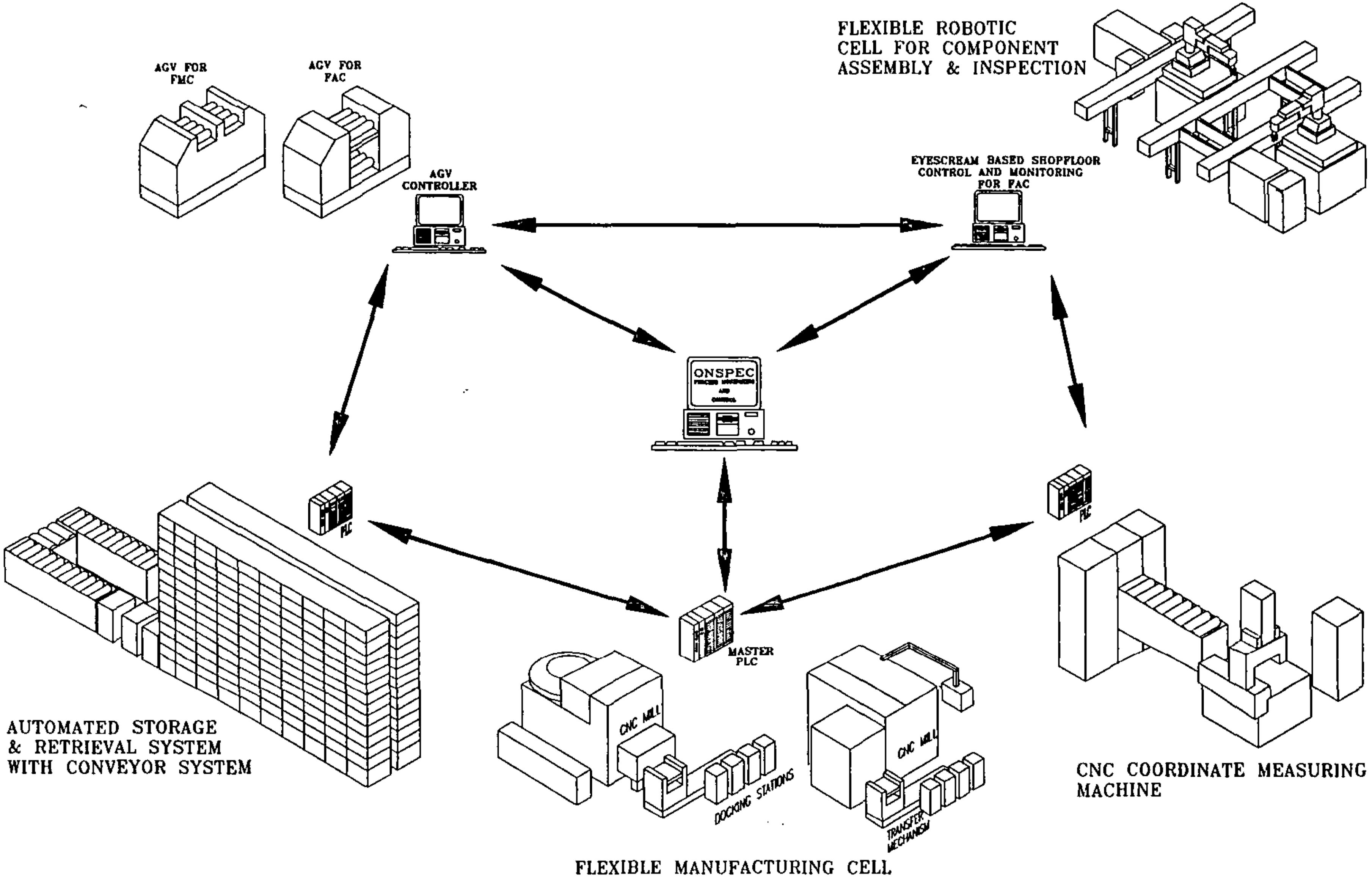
- Distribution of activities within the CIM hierarchy

CIM MODEL
SHOP FLOOR LAYOUT



ANNEX II

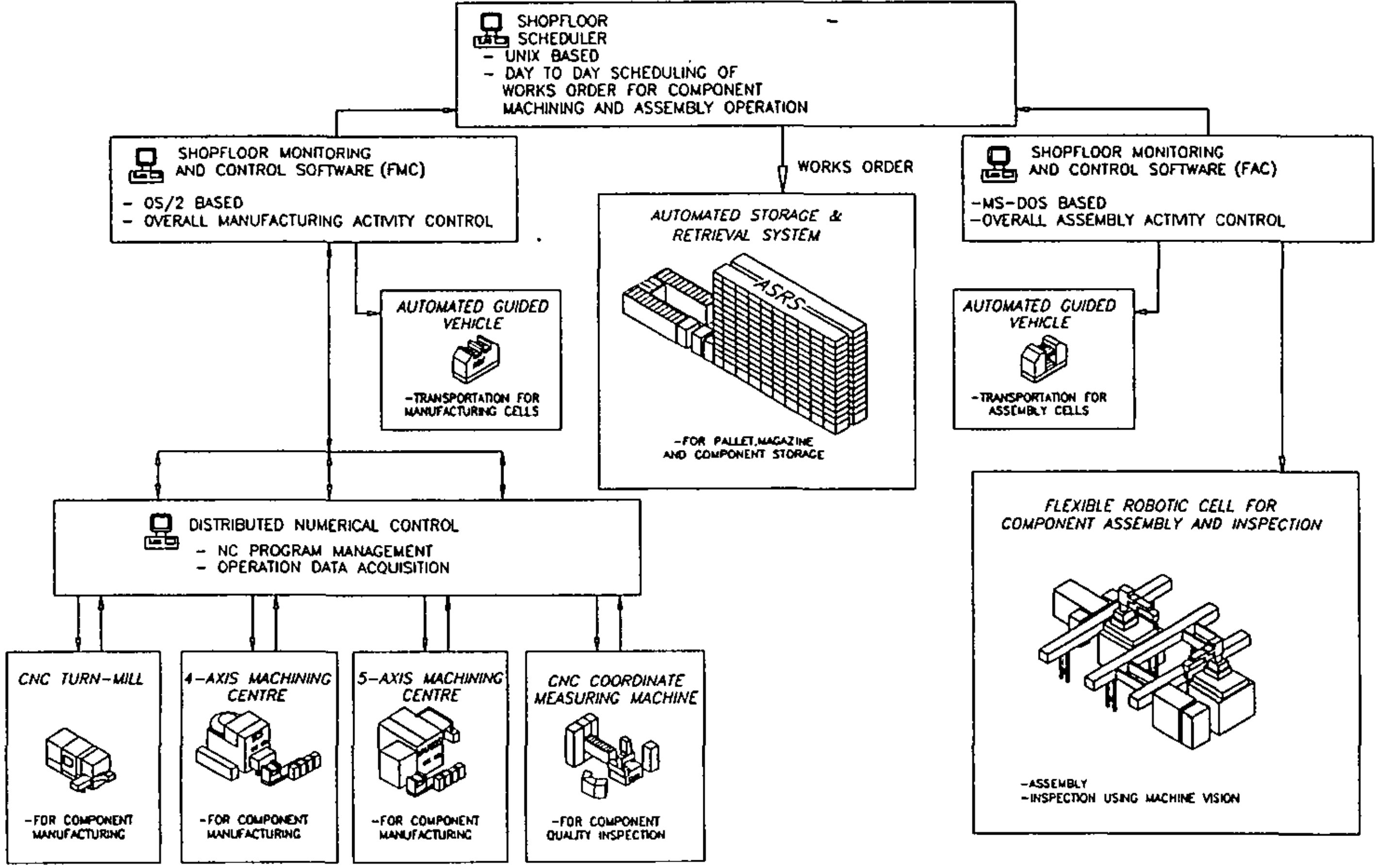
SHOPFLOOR MONITORING AND CONTROL


AGV FOR FMC
AGV FOR FAC
AGV CONTROLLER
FLEXIBLE ROBOTIC CELL FOR COMPONENT ASSEMBLY & INSPECTION
EYESCREAM BASED SHOPFLOOR CONTROL AND MONITORING FOR FAC
ONSPEC
PLC
PLC
MASTER PLC
AUTOMATED STORAGE & RETRIEVAL SYSTEM WITH CONVEYOR SYSTEM
CNC MILL
DOCKING STATIONS
CNC MILL
TRANSFER MECHANISM
FLEXIBLE MANUFACTURING CELL
CNC COORDINATE MEASURING MACHINE

ANNEX III

ANNEX IV

CIM ARCHITECTURE FOR SHOPFLOOR CONTROL & MONITORING



SHOPFLOOR SCHEDULER
- UNIX BASED
- DAY TO DAY SCHEDULING OF WORKS ORDER FOR COMPONENT MACHINING AND ASSEMBLY OPERATION

SHOPFLOOR MONITORING AND CONTROL SOFTWARE (FMC)
- OS/2 BASED
- OVERALL MANUFACTURING ACTIVITY CONTROL

WORKS ORDER

AUTOMATED STORAGE & RETRIEVAL SYSTEM
-FOR PALLET,MAGAZINE AND COMPONENT STORAGE

SHOPFLOOR MONITORING AND CONTROL SOFTWARE (FAC)
-MS-DOS BASED
-OVERALL ASSEMBLY ACTIVITY CONTROL

AUTOMATED GUIDED VEHICLE
-TRANSPORTATION FOR MANUFACTURING CELLS

AUTOMATED GUIDED VEHICLE
-TRANSPORTATION FOR ASSEMBLY CELLS

DISTRIBUTED NUMERICAL CONTROL
- NC PROGRAM MANAGEMENT
- OPERATION DATA ACQUISITION

FLEXIBLE ROBOTIC CELL FOR COMPONENT ASSEMBLY AND INSPECTION
-ASSEMBLY
-INSPECTION USING MACHINE VISION

CNC TURN-MILL
-FOR COMPONENT MANUFACTURING

4-AXIS MACHINING CENTRE
-FOR COMPONENT MANUFACTURING

5-AXIS MACHINING CENTRE
-FOR COMPONENT MANUFACTURING

CNC COORDINATE MEASURING MACHINE
-FOR COMPONENT QUALITY INSPECTION

APPENDIX III

INFORMATION MODELS

Manufacturing Facility
Part Master/ BOM
Resource
Process Plan
Order Entry
Schedule
WIP
Engineering Resource
Manufacturing Cell
Customer
Supplier

Schedule

FIELD DEFINITION	SIZE	DESCRIPTION
Manufacturing order number	N[7]	User defined unique identifier for a batch of manufacturing order.
Part number	C[15]	Unique identifier for the part (or component).
Priority	N[3]	It sorts and dictates the sequencing of scheduled jobs.
Order Status	C[1]	It indicates the part's current position within the order process.
Schedule Status	C[10]	It indicates the status of the part within the scheduling & manufacturing process.
Planned Quantity	N[9]	It specifies how many or how much of the part is required for manufacture.
Unit of Measure	C[4]	It is the standard quantitive unit for the part used in the manufacturing process.
Schedule Start Date	Date	Planned date of commencement for the manufacture of the required quantity of parts..
Schedule End Date	Date	Planned date of completion for the manufacture of the required quantity of parts.

(Schedule / Unschedule / Complete / WIP / Hold)

2 = Quotation or Firmed planned forecast

3 = Open Order. Confirmed order but all required issues or shipment have not been made for the item.

4 = Released Order. Confirmed and planned but pending receipt of required issues or shipments.

5 = Closed Order. All required issues or shipment have been made for the item and can therefore proceed with manufacture.

6 = Closed Order. Manufacture of the item is complete and is ready for dispatch.

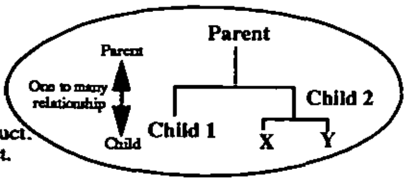
7 = Closed Order. The order is ready to be deleted from the active file and retained in order history.

8 = Closed Order. Purge the order and do not retain in order history.

9 = Credit Hold. The customer's credit limit has been exceeded or the order is placed on hold for another reason. The item is then treated as an open order.

Part Master/ BOM

The bill of materials (BOM) is a list of the items, ingredients or materials needed to produce a parent item, end item, or product. It is not just a simple listing of dependent demand items, but a structured list which describes the sequence of steps in manufacturing the product.



The BOM is utilised for the following :

- Manufacturing engineering use the BOM to show how to manufacture the product.
- Production planning use the BOM to schedule the parts which make the product.
- Manufacturing use the BOM to make the product on the shop floor.
- Finance use the BOM to cost the product.
- Order entry use the BOM to translate customer orders and enquiries based upon the inter-dependency and relationships of components! for the product in terms of manufacture, procurement and etc.

	FIELD DEFINITION	SIZE	DESCRIPTION
Part Master/Parent	**Parent Part Number**	C[15]	Unique identifier for the parent item which is the higher level item in the BOM.
	Effectivity start Date	Date	Date on which the part is introduced into the BOM.
	Effectivity End Date	Date	Date on which the part is severed from the BOM
	Unit of Measure	C[4]	It is the standard quantitive unit for the part used in the manufacturing process.
	Engineering Change Notice Number	N[7]	The revision for the BOM as authorized by the engineering department.
	Change Effected by	C[20]	Personnel, section or department responsible for the change.
	Date	Date	Date on which the engineering revision was initiated.
	Phases out Part Number	C[15]	Preceeding part that was in use.
	Phased out by Part Number	C[15]	Succeeding part to be used.
	Number of levels	N[2]	Number of levels in BOM supported for the parent item to be produced.
	Number of children	N[3]	Number of different sub-components supported at the specified level.
BOM Child	**Parent Part Number**	C[15]	Unique identifier for the parent item which is the higher level item in the BOM.
	Number of Children	N[3]	Number of different sub-components supported at the specified level.
	Part Number (Child)	C[15]	Unique identifier for the part (or component).
	Part Type	C[1]	It distinguishes the various types of relationship between a component & its parent item in the BOM.
	Quantity per assembly	N[12,2]	Number of components required for assembly of per unit parent item.
	Effectivity start Date	Date	Date on which the part is introduced into the BOM.
	Effectivity End Date	Date	Date on which the part is severed from the BOM
	Unit of Measure	C[4]	It is the standard quantitive unit for the part used in the manufacturing process.
	Lead-time offset	N[9]	It is the difference between the due date and the release date.
	Engineering Change Notice Number	N[7]	The revision for the BOM as authorized by the engineering department.
	Change Effected by	C[20]	Personnel, section or department responsible for the change.
	Date	Date	Date on which the engineering revision was initiated.
	Phases out Part Number	C[15]	Preceeding part that was in use.
	Phased out by Part Number	C[15]	Succeeding part to be used.

N = Normal component that is consumed in the manufacture of its parent.

P = Phantom component that is used for BOM structuring purposes only (e.g. a transient subassembly consumed in the manufacture of its parent.

R = Resource component used in the planning process of the manufacture of its parent (e.g. labour & machining hrs).

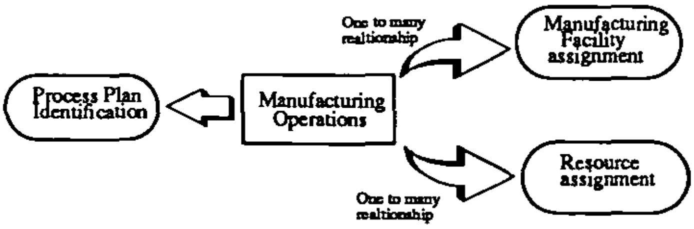
C = Co-product component derived from the manufacture of the parent.

T = Tool component used in the manufacture of the part.

U = Tool return item which will be returned after manufacture of the part.

Process Plan

The process plan will involve assignment of manufacturing facilities and resources to each of the planned manufacturing operation.. The process routing is geared towards cellular type of production where Group Technology is applied to identify "sameness" of parts, equipments or processes in order to derive suitable working cells.



	FIELD DEFINITION	SIZE	DESCRIPTION
Process Plan Identification	Part Number	C[15]	Unique identifier for the part (or component).
	Process Plan ID	N[7]	Unique identifier for the process plan.
	Process Description	C[60]	Brief textual description pertaining to the process routing.
	Manufacturing Cell assignment	N[3]	Unique identifier for assigned manufacturing cell where part need to be delivered. for manufacture.
Manufacturing Operation Assignment	Process Plan ID	N[7]	Unique identifier for the process plan.
	Manufacturing Operation ID	N[7]	Unique identifier for the manufacturing operation.
	Manufacturing Operation Description	C[60]	Brief textual description.
	Preceeding Manufactulrng Operation ID	N[7]	Unique identifier for preceeding manufacturing operation.
	Next Manufacturing Operation ID	N[7]	Unique identifier for next manufacturing operation to be performed.
	Alternative Manufacturing Operation ID	N[7]	Unique identifier for alternative manufacturing operation.
	Setup time per unit item (min)	N[6,2]	Time required to equip and prepare the "work centre" or cell for production.
	Machining time per unit item (min)	N[6,2]	Actual productive time for manufacture of part.
	Handling time per unit item (min)	N[6,2]	Time required for handling of part which includes transportation.
	Operation time (min)	N[10,2]	Cumulative setup, machining and handling times for the manufacture of the order.
	Scrap rate	N[5,2]	The percentage difference between the amount or number of unit parts started in a manufacturing process and that amount or number of units which is completed at an acceptable quality level.
Manufacturing Facility Assignment	Manufacturing Operation ID	N[7]	Unique identifier for the manufacturing operation.
	Asset ID	N[7]	Unique identifier for either an employed personnel or an item which is owned by the business and has value that can be measured objectively.
	Feed	N[4]	Machining feedrate.
	Speed (mm/min)	N[6]	Cutting speed.
	Depth of cut (mm)	N[3]	
	Number of passes	N[5]	
	Remarks	C[60]	Textual comment.
Resource Assignment	Manufacturing Operation ID	N[7]	Unique identifier for the manufacturing operation.
	Resource ID	C[15]	Unique identifier assigned to resource item

	Resource type	C[2]	Classification of resources :
			T = Toolings
			TA = Tooling Accessories or Attachments
			M = Materials
			F = Fixturing elements
			FA = Fixturing Accessories or Attachments
			M = Miscellaneous

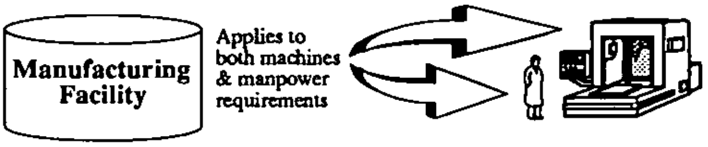
	Quantity required	N[12,2]	Total quantity of resource item to be allocated or reserved.
	Unit of Measure	C[4]	It is the standard quantitive unit for the part used in the manufacturing process.

WIP

FIELD DEFINITION	SIZE	DESCRIPTION
Manufacturing Order number	N[7]	User defined unique identifier for a batch of manufacturing order.
Part Number	C[15]	Unique identifier for the part (or component).
Actual Quantity produced	N[9]	The number of parts manufactured.
Work centre or Cell utilization rate	N[5,2]	The percent time that a "work centre" or cell is running production.
Actual capacity (hrs)	N[5,2]	Capacity calculated from actual performance data, i.e. number ofparts produced multiplied by standard hrs per part.

Order Entry

FIELD DEFINITION	SIZE	DESCRIPTION
Customer ID	N[7]	Unique identifier assigned to a customer.
Preceeding Manufacturing Order Number	N[7]	Unique identifier assigned for the previous batch of manufacturing order.
Current Manufacturing Order Number	N[7]	Unique identifier assigned for the current batch of manufacturing order.
Parent Part Number	C[15]	Unique identifier for the parent item which is the higher level item in the BOM.
Description	C[60]	Brief textual description of the customer order/product.
Product Effectivity Start Date	Date	Date from which the product is being supported.
Product Effectivity End Date	Date	Date from which the product is no longer supported.
Type (Repeat/One Off)	C[1]	Classification of manufacturing order for either repeat or one off type.
Due Date	Date	Planned completion or shipment date for the product.
Unit of Measure	C[4]	It is the standard quantitive unit for the part used in the manufacturing process.
Unit Price	N[12,2]	It is the price per unit of the item being ordered.
Order Quantity	N[9]	Number of items ordered at the specified unit of measure.



	FIELD DEFINITION	SIZE	DESCRIPTION
	Asset ID	N[7]	Unique identifier for either a personnel or an item which is owned by the business and has value that can be measured objectively.
	Description	C[60]	Brief textual description for the item.
	Location	C[15]	Physical location where the personnel is assigned or where the asset item can be found or is being currently used.
	Working capacity (hrs)	N[3]	Allocated productive time available for working.
	Labour cost / hr	N[7,2]	
	Handling cost / hr	N[7,2]	
Machine & Work parameters	Asset ID	N[7]	
	Supplier ID	N[7]	Unique identifier for supplier of asset item.
	Last Service / Maintenance Date	Date	Date on which service or maintenance is carried out on the asset item.
	Repaired on	Date	Date on which repair work was carried out on the asset item.
	Repair work order number	N[7]	Work order number for repair work on the asset item.
	Maximum job size accommodated		Physical size of part that can be handled without any problems by the asset item.
	X axis (mm)	N[5,2]	
	Y axis (mm)	N[5,2]	
	Z axis (mm)	N[5,2]	
	Accuracy	N[4,2]	The degree of freedom from error.
	Standard Cost		
	Machining cost / hr	N[7,2]	
	Utilities		
	Horse power	N[7]	
	Speed range Maximum (mm/min)	N[6]	
	Minimum (mm/min)	N[6]	
	Feed range Maximum (mm/min)	N[4]	
	Minimum (mm/min)	N[4]	
	Payload (kg)	N[5]	
	Work envelope		
	X axis (mm)	N[5,2]	
	Y axis (mm)	N[5,2]	
	Z axis (mm)	N[5,2]	
	A axis (mm)	N[5,2]	
	B axis (mm)	N[5,2]	
	Standard Times		
	Setup time (min)	N[5,2]	
	Tool change time (min)	N[5,2]	
	Feed change time (min)	N[5,2]	
	Speed change time (min)	N[5,2]	
	Table rotation time (min)	N[5,2]	
	Tool adjustment time (min)	N[5,2]	
	Rapid tranverse (mm/min)	N[5,2]	
Personnel	Asset ID	N[7]	
	Personnel ID	C[15]	
	Name	C[30]	
	Salary	N[7,2]	
	Address	C[60]	
	Telephone	C[20]	
	Skill	C[30]	Skills or expertise personnel possess.
	Skill level	N[2]	
	Remarks	C[60]	

Resource ☞ Applies to Tools/Materials/Fixtures

FIELD DEFINITION	SIZE	DESCRIPTION
Resource ID	C[15]	Unique identifier assigned to resource item

Resource type	C[2]	Classification of resources :
		T = Toolings
		TA = Tooling Accessories or Attachments
		M = Materials
		F = Fixturing elements
		FA = Fixturing Accessories or Attachments
		M = Miscellaneous

Description	C[60]	Brief textual description for resource item.
Location	C[15]	Storage or usage area where the resource item can be found.
Account Number	N[15]	Assigned number for purchase of the item.
Unit of Measure	C[4]	It is the standard quantitive unit for the part used in the manufacturing process.
Unit Price	N[12,2]	It is the price per unit of the item being ordered.
Buy/Make/Supply Code	C[1]	It indicates if the item is as follows : M = Make (Manufactured in-house) B = Buy (Purchased and no parts need to be supplied to the vendor) S = Supplied (Purchased but supplied to the vendor)
Supplier ID	N[7]	Unique identifier assigned to the supplier of the resource item.
Catalogue Order Number	C[30]	Catalog number for the supplied resource item.
Purchasing Lead Time	N[7]	The span of time required to obtain a purchased item which includes procurement lead time, vendor lead time, transportationtime, receiving, inspection and put away time.
Last Order Date	Date	Date on which the last order was placed for the resource item.
Quantity Ordered	N[9]	Number of items ordered.
Effectivity Start Date	Date	Date from which the resource item is being supported.
Effectivity End Date	Date	Date from which the resource item is no longer supported.
Stock on-hand	N[9]	Physical stock on-hand minus allocations, reservations and (usually) quantities held for quality problems.
Allocated/Reserved Stock	N[9]	Commited resource item.
Scrap value	N[7,2]	Value of scrap per unit measure of scrap.
Unit of measure for scrap	C[3]	It is the standard quantitive unit for the scrap item.

Manufacturing Cell

☞ Grouping of manufacturing stations into cells for part manufacture.

FIELD DEFINITION	SIZE	DESCRIPTION
Manufacturing Cell Group ID	N[2]	Unique identifier for manufacturing cell.
Number of manufacturing stations	N[2]	Number of manufacturing stations or processes supported in the configuration.
Manufacturing Station 1 - Assest ID	N[7]	Unique identifier for manufacturing station.
Description	C[60]	Brief textual description for manufacturing activity.
Manufacturing Station 2 - Asset ID	N[7]	- ditto -
Description	C[60]	
Manufacturing Station 3 - Asset ID	N[7]	- ditto -
Description	C[60]	
Manufacturing Station 4 - Asset ID	N[7]	- ditto -
Description	C[60]	
Manufacturing Station 5 - Asset ID	N[7]	- ditto -
Description	C[60]	

Customer

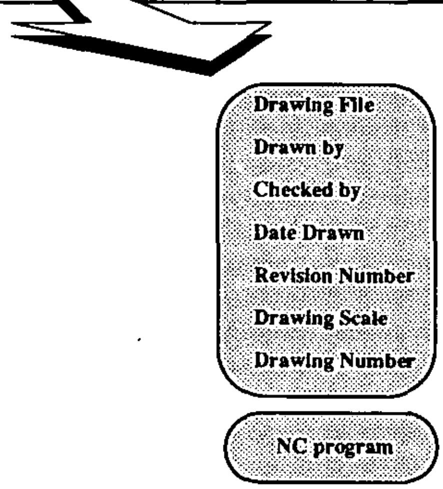
FIELD DEFINITION	SIZE	DESCRIPTION
Customer ID	N[7]	Unique identifier assigned to a customer.
Company/Name	C[40]	Name of customer.
Address	C[60]	
Contact Person	C[25]	
Telephone	C[20]	
Fax	C[20]	

Supplier

FIELD DEFINITION	SIZE	DESCRIPTION
Supplier ID	N[7]	Unique identifier assigned to a supplier.
Company/Name	C[40]	Name of supplier.
Address	C[60]	
Contact Person	C[25]	
Telephone	C[20]	
Fax	C[20]	

Engineering
Resource

FIELD DEFINITION	SIZE	DESCRIPTION
Part Number	C[15]	Unique identifier for the part (or component).
Location	C[10]	
Engineering Resource	C[10]	

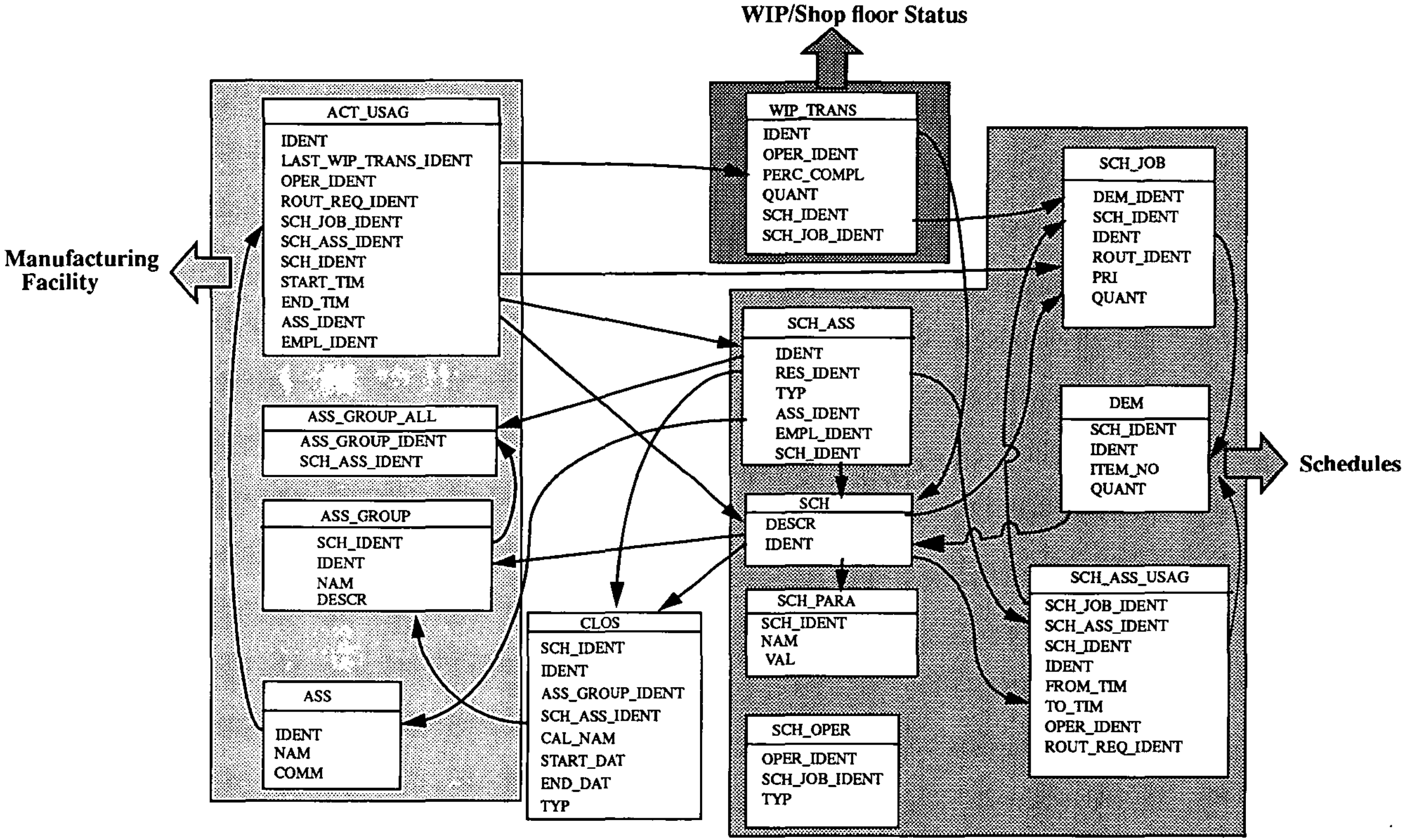


Drawing File
Drawn by
Checked by
Date Drawn
Revision Number
Drawing Scale
Drawing Number

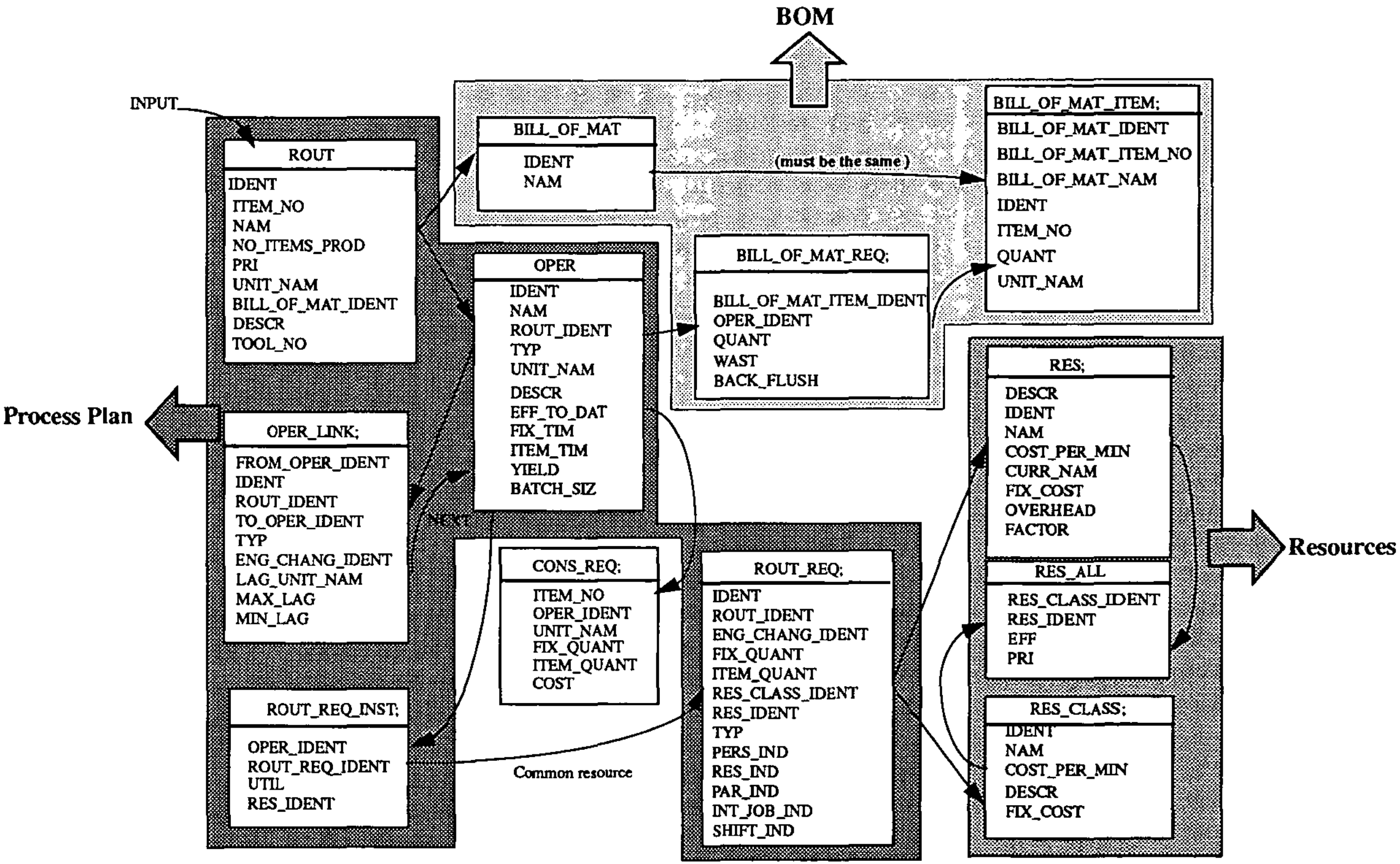
NC program

APPENDIX IV

Association between information models and information representations in MCC and ELMS CAPM packages



SCHEDULE DATA TABLES IN MCC



ROUT
- IDENT
- ITEM_NO
- NAM
- NO_ITEMS_PROD
- PRI
- UNIT_NAM
- BILL_OF_MAT_IDENT
- DESCR
- TOOL_NO

BILL_OF_MAT
- IDENT
- NAM

BILL_OF_MAT_ITEM;
- BILL_OF_MAT_IDENT
- BILL_OF_MAT_ITEM_NO
- BILL_OF_MAT_NAM
- IDENT
- ITEM_NO
- QUANT
- UNIT_NAM

OPER
- IDENT
- NAM
- ROUT_IDENT
- TYP
- UNIT_NAM
- DESCR
- EFF_TO_DAT
- FIX_TIM
- ITEM_TIM
- YIELD
- BATCH_SIZ

BILL_OF_MAT_REQ;
- BILL_OF_MAT_ITEM_IDENT
- OPER_IDENT
- QUANT
- WAST
- BACK_FLUSH

OPER_LINK;
- FROM_OPER_IDENT
- IDENT
- ROUT_IDENT
- TO_OPER_IDENT
- TYP
- ENG_CHANG_IDENT
- LAG_UNIT_NAM
- MAX_LAG
- MIN_LAG

CONS_REQ;
- ITEM_NO
- OPER_IDENT
- UNIT_NAM
- FIX_QUANT
- ITEM_QUANT
- COST

ROUT_REQ;
- IDENT
- ROUT_IDENT
- ENG_CHANG_IDENT
- FIX_QUANT
- ITEM_QUANT
- RES_CLASS_IDENT
- RES_IDENT
- TYP
- PERS_IND
- RES_IND
- PAR_IND
- INT_JOB_IND
- SHIFT_IND

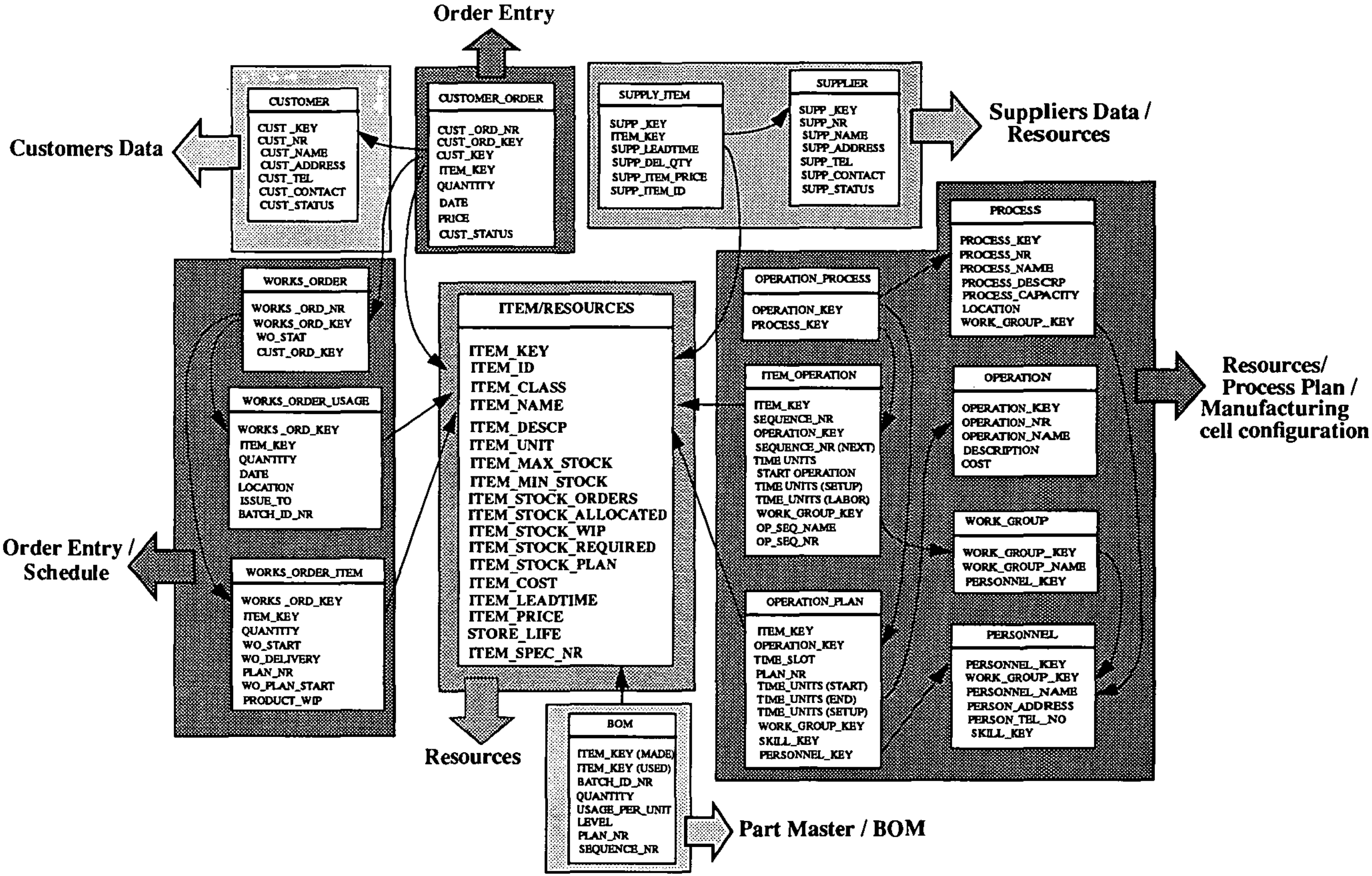
ROUT_REQ_INST;
- OPER_IDENT
- ROUT_REQ_IDENT
- UTIL
- RES_IDENT

RES;
- DESCR
- IDENT
- NAM
- COST_PER_MIN
- CURR_NAM
- FIX_COST
- OVERHEAD
- FACTOR

RES_ALL
- RES_CLASS_IDENT
- RES_IDENT
- EFF
- PRI

RES_CLASS;
- IDENT
- NAM
- COST_PER_MIN
- DESCR
- FIX_COST

PRODUCT & PROCESS DATA TABLES IN MCC



DATA RELATIONSHIPS IN ELMS

APPENDIX V

Database 'Driver'

Services offered

Database Driver	
Services Offered	**Details**
CONNECT	Provides a direct connection to the database and loads the definition of data access objects from a file. An application must connect to the database before it can access any data. The parameters for the connect service are the username and password to establish the database connection and the file name from which the definition of the data access objects is loaded.
DISCONNECT	Closes the connection to the database, established by the connect service. Before the connection is closed, the changes performed in the current transaction are discarded. To make the changes permanent, the commit service must be used. The disconnect service requires no arguments.

SELECT	Retrieves rows and columns from one or more SQL tables. When requesting the select service, a data access object must be specified. Optional arguments are an additional search condition and a file name. The actual SQL statement is build up using the definition of the specified data access object and the additional search condition if present. Assuming that the data access object has a search condition defined and an additional search condition is given when the select service is requested, the SQL statement has the following form:
	SELECT object field list FROM object tables
	WHERE object search condition AND additional search condition
	The retrieved data is written to a file, specified by the file name when the service was requested. If no file name is given, the result is written to a global character string. Because of the fixed length of the character string it should only be used, when the amount of expected data doesn't exceed this length. Otherwise the rest of the data is discarded. When an error occurs while the SQL statement is executed, this is also indicated in the global character string and the specified file will be empty.

INSERT	Adds new rows to a database table. The insert service requires two mandatory arguments. One to specify a data access object, the SQL statement should refer to and the second argument to define the data that is to be inserted in the database table. The insert service can only be requested for data access objects that refer to one table in the table definition. Otherwise an error will occur when executing the SQL statement. The SQL statement has the following form:
	INSERT INTO object table (object field list)
	VALUES (new data)
	A further restriction of data access objects that can be used for the insert service concerns the list of field names. In this list, all fields must be present that are forced to contain a value by the definition of the database table, i.e. all fields, specified as NOT NULL. The new data argument must contain a value for each field name given in the same order as in the data access object definition. The result of the execution of the SQL statement will be indicated in a global character string.

Continue..... ☞

Database Driver	
Services Offered	**Details**

UPDATE	Changes the data in a table. The update service has two mandatory and one optional argument. The first mandatory argument specifies a data access object and the second gives field names and their new values. In the optional argument, an additional search condition can be specified. The update service is again restricted to data access objects which refer to only one database table. If an additional search condition was given when the update service was requested and the specified data access object has a search condition defined, the actual SQL statement has the following form:
	UPDATE object table SET change data
	WHERE object search condition AND additional search condition
	The actual number of changed rows will be indicated in a global character string. It is not treated as an error if no row is changed.

DELETE	Removes rows from a table. When requesting the delete service, a data access object must be specified. An additional search condition can be given, but is not mandatory. The delete service can only be requested for data access objects, referring to one database table. Otherwise an error will be indicated. If an additional search condition is given and the specified data access object has a search condition defined. The actual SQL statement has the following form: DELETE FROM object table WHERE object search condition AND additional search condition. The number of deleted rows will be indicated in a global character string. It is not treated as an error if no row is deleted. If one of the optional search conditions is not present in the services select, update or delete, the where clause will have the following from: WHERE single search condition If none of both search conditions is defined, no where clause is added to the actual SQL statement.
COMMIT	Makes permanent all changes performed in the current transaction (naturally, the data may be changed by future updates). Before the commit service is requested, the changes performed in the current transaction are not visible for other users of the same database. The commit service marks the end of the current transaction and the beginning of a new one. No arguments are required for this service.
ROLLBACK	Undo the work done in the current transaction. The rollback service requires no arguments.

Program listings

```
#include <stdio.h>
#include “local_incl.h”

FILE *temp_file;
char column[40];
char file_line[132];
char file_name[60];

extern char result_string[VERY_LONG_STRING];

main()
{
 int i;

 strcpy(file_name, “/home2/sandra/valdew/oracle_c/progs/example_1_tem”);

 mw_connect(“mcc21”, “m”, “/home2/sandra/valdew/oracle_c/progs/file_1.txt”);

/* Usage of Files */
mw_query(“object1”, ““, file_name);
 if((temp_file = fopen(file_name,”r”)) == NULL)
printf(“\nError, open.”);
 file_line[0] = ‘\0’;
 i = 1;
while(file_reader(column, temp_file, file_line))
 {
printf(“\n%d. Value: %s.”,i++,column);
 }
 fclose(temp_file);


/* Usage of Query */

mw_query(“object1”,”item_no = ‘LUT-B1’”,””);
printf(“\nResult_String: %s.”,result_string);
i = 1;
while(next_value(result_string, column, result_string))
 {
printf(“\n%d. Value: %s.”,i++,column);
printf(“\nResult_String: %s.”,result_string);
 }


/* Usage of insert */

mw_insert(“object1”, “‘Example access_ora’, ‘Example’”);
printf(“\nResult_String: %s.”,result_string);
mw_query(“object2”,”item_no = ‘Example’”,””);
printf(“\nResult_String: %s.”,result_string);
i = 1;
while(next_value(result_string, column, result_string))
 {
printf(“\n%d. Value: %s.”,i++,column);
printf(“\nResult_String: %s.”,result_string);
 }
/* Usage of update */

mw_update(“object1”, “item_descr = ‘Ex’”, “”);
printf(“\nResult_String: %s.”,result_string);
mw_query(“object1”,””,””);
printf(“\nResult_String: %s.”,result_string);
i = 1;
while(next_value(result_string, column, result_string))
 {
printf(“\n%d. Value: %s.”,i++,column);
printf(“\nResult_String: %s.”,result_string);
 }


/* Usage of delete */

mw_delete(“object1”, “item_no = ‘LUT-B1’”);
printf(“\nResult_String: %s.”,result_string);
mw_query(“object1”,”item_no = ‘LUT-B1’”,””);
printf(“\nResult_String: %s.”,result_string);
i = 1;
while(next_value(result_string, column, result_string))
 {
printf(“\n%d. Value: %s.”,i++,column);
printf(“\nResult_String: %s.”,result_string);
 }



 printf(“\n”);
/*mw_commit(); mw_rollback(); */
 mw_disconnect();
}
```

Meta-file definition

Name of meta-file : file_1.txt

Definition of database-objects: (for access_ora)

"object1" object name "item i, bill_of_mat b" table name
"i.item_descr, b.item_no, b.nam" columns "i.item_no = b.item_no (+) AND i.item_no LIKE
'LUT-B1%'" where clause

"object2" object name "item" table name
"item_descr, item_no" columns "" where clause

Program library functions for embedded SQL command calls to perform required 'Driver' services

```
/* File name & Package Name */
struct sqlcxp
{
 unsigned short fillen;
 char filnam[13];
};
static struct sqlcxp sqlfpn =
{
 13,
 "access_ora.pc"
};


static unsigned long sqlctx = 0;


static struct sqlexd {
 unsigned long sqlvsn;
 unsigned short arrsiz;
 unsigned short iters;
 unsigned short offset;
 unsigned short selerr;
 unsigned short sqlety;
 unsigned short unused;
 short *cud;
 unsigned char *sqlest;
 char *stmt;
 unsigned char **sqphsv;
 unsigned long *sqphsl;
 short **sqpind;
 unsigned char *sqhstv[3];
 unsigned long sqhstl[3];
 short *sqindv[3];
} sqlstm = {1,3};
extern sqlcex(/*_ unsigned long *, struct sqlexd *, struct sqlcxp * _*/);
extern sqlbuf(/*_ char * _*/);
extern sqlora(/*_ long *, void * _*/);

static int IAPSUCC = 0;
static int IAPFAIL = 1403;
static int IAPFTL = 535;
extern sqliem();
/* cud (compilation unit data) array */
static short sqlcud0[] =
{1,2,
2,0,0,0,27,192,3,3,0,1,0,1,5,0,0,1,5,0,0,1,10,0,0,
25,0,0,0,29,461,0,0,0,1,0,
36,0,0,0,31,480,0,0,0,1,0,
47,0,1,5,17,545,1,1,0,1,0,1,5,0,0,
62,0,1,0,19,556,1,0,0,1,0,0,32,0,0,
77,0,1,0,11,576,1,0,0,1,0,0,32,0,0,
92,0,1,0,20,586,1,0,0,1,0,0,32,0,0,
107,0,1,0,14,714,1,0,0,1,0,0,32,0,0,
122,0,1,0,31,794,0,0,0,1,0,
133,0,1,0,15,837,0,0,0,1,0,
144,0,1,0,32,838,0,0,0,1,0,
155,0,1,10,17,888,1,1,0,1,0,1,5,0,0,
170,0,2,0,24,891,1,1,0,1,0,1,5,0,0,
185,0,2,0,31,915,0,0,0,1,0,
196,0,1,10,17,971,1,1,0,1,0,1,5,0,0,
211,0,3,0,24,974,1,1,0,1,0,1,5,0,0,
226,0,3,0,31,998,0,0,0,1,0,
237,0,1,10,17,1051,1,1,0,1,0,1,5,0,0,
252,0,4,0,24,1054,1,1,0,1,0,1,5,0,0,
267,0,4,0,31,1078,0,0,0,1,0,
};

/***************************************************************************/
/* File: access_ora.pc */
/* Function: The functions in this file provide access to an oracle database */
/* using embedded SQL-statements. */
/* First, the programm has to connect to the oracle database using the */
/* function mw_connect. Then the data can be accessed and manipulated by */
/* the functions mw_query, mw_insert, mw_update and mw delete. To make */
/* changes to the database permanent, the function mw_commit has to be */
/* called. To discard changes, call the mw_rollback function. The function */
/* mw_disconnect is used to end the database session. Note that mw_disconnect */
/* discards all changes that have not been commited. */
/***************************************************************************/

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "local_incl.h"

#ifndef NULL
#define NULL 0
#endif



/***************************************************************************/
/* Only for debuging purpos */
/***************************************************************************/

#define NOPRINT 0
#define NOSTATEMENTS 0

/***************************************************************************/
/* Maximum number of select-list items, allowed in an select statement. */
/***************************************************************************/

#define MAX_ITEMS 30

/***************************************************************************/
/* Maximum lengths of the names of the select-list items in an select */
/* statement. */
/***************************************************************************/
```

```
#define MAX_VNAME_LEN 30

/*****************************************************************************/
/* This string is used to process the SQL-statements */
/*****************************************************************************/

/* SQL stmt #1
EXEC SQL BEGIN DECLARE SECTION;
*/
 char sql_statement[1024];
/* SQL stmt #2
 EXEC SQL VAR sql_statement IS STRING(1024);
EXEC SQL END DECLARE SECTION;
*/

/* SQL stmt #4
EXEC SQL INCLUDE sqlca;
*/
/*
 * $Header: sqlca.h,v 1040100.1 91/02/26 00:14:09 epotteng Generic<base> $ sqlca.h
 */

/* Copyright (c) 1985,1986 by Oracle Corporation. */

/*
NAME
 SQLCA : SQL Communications Area.
FUNCTION
 Contains no code. Oracle fills in the SQLCA with status info
 during the execution of a SQL stmt.
NOTES
 If the symbol SQLCA_STORAGE_CLASS is defined, then the SQLCA
 will be defined to have this storage class. For example:

 #define SQLCA_STORAGE_CLASS extern

 will define the SQLCA as an extern.

If the symbol SQLCA_INIT is defined, then the SQLCA will be statically initialized. Although this is not necessary in order to use the SQLCA, it is a good pgming practice not to have unitialized variables. However, some C compilers/OS's don't allow automatic variables to be init'd in this manner. Therefore, if you are INCLUDE'ing the SQLCA in a place where it would be an automatic AND your C compiler/OS doesn't allow this style of initialization, then SQLCA_INIT should be left undefined -- all others can define SQLCA_INIT if they wish.

MODIFIED
 Clare 12/06/84 - Ch SQLCA to not be an extern.
 Clare 10/21/85 - Add initialization.
 Bradbury 01/05/86 - Only initialize when SQLCA_INIT set
 Clare 06/12/86 - Add SQLCA_STORAGE_CLASS option.
*/

#ifndef SQLCA
#define SQLCA 1
```

```
struct sqlca
 {
/* ub1 */ char sqlcaid[8];
/* b4 */ long sqlabc;
/* b4 */ long sqlcode;
 struct
 {
/* ub2 */ unsigned short sqlerrml;
/* ub1 */ char sqlerrmc[70];
 } sqlerrm;
/* ub1 */ char sqlerrp[8];
/* b4 */ long sqlerrd[6];
/* ub1 */ char sqlwarn[8];
/* ub1 */ char sqlext[8];
 };

#ifdef SQLCA_STORAGE_CLASS
SQLCA_STORAGE_CLASS struct sqlca sqlca
#else
 struct sqlca sqlca
#endif

#ifdef SQLCA_INIT
 = {
 {‘S’, ‘Q’, ‘L’, ‘C’, ‘A’, ‘ ‘, ‘ ‘, ‘ ‘},
 sizeof(struct sqlca),
 0,
 { 0, {0}},
 {‘N’, ‘O’, ‘T’, ‘ ‘, ‘S’, ‘E’, ‘T’, ‘ ‘},
 {0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0}
 }
#endif
 ;

#endif

/* end SQLCA */
/* SQL stmt #5
EXEC SQL INCLUDE sqlda;
*/
/*
 * $Header: sqlda.h,v 1040100.1 91/02/26 00:14:15 epotteng Generic<base> $ sqlda.h
 */

/**************************************************************
* The SQLDA descriptor definition *
*--------------------------------------------------------------*
* VAX/3B Version *
* *
* Copyright (c) 1987 by Oracle Corporation *
**************************************************************/
```

```
/* MODIFIED
 Morse 12/01/87 - undef L and S for v6 include files
 Richey 07/13/87 - change int defs to long
 Clare 09/13/84 - Port: Ch types to match SQLLIB structs
 Clare 10/02/86 - Add ifndef SQLDA
*/

#ifndef SQLDA_
#define SQLDA_ 1

#ifdef T
# undef T
#endif
#ifdef F
# undef F
#endif

#ifdef S
# undef S
#endif
#ifdef L
# undef L
#endif

struct SQLDA {
 long N; /* Descriptor size in number of entries */
 char **V; /* Ptr to Arr of addresses of main variables */
 long *L; /* Ptr to Arr of lengths of buffers */
 short *T; /* Ptr to Arr of types of buffers */
 short **I; /* Ptr to Arr of addresses of indicator vars */
 long F; /* Number of variables found by DESCRIBE */
 char **S; /* Ptr to Arr of variable name pointers */
 short *M; /* Ptr to Arr of max lengths of var. names */
 short *C; /* Ptr to Arr of current lengths of var. names */
 char **X; /* Ptr to Arr of ind. var. name pointers */
 short *Y; /* Ptr to Arr of max lengths of ind. var. names */
 short *Z; /* Ptr to Arr of cur lengths of ind. var. names */
 };

typedef struct SQLDA SQLDA;

#endif

SQLDA *bind_dp;
SQLDA *select_dp;

/****************************************************************************/
/* Functions external to this module. */
/****************************************************************************/

extern SQLDA *sqlald();
extern void sqlnul();


/****************************************************************************/
/* A global flag for the SQL error routine. */
/****************************************************************************/
```

```
int parse_flag = 0;

/****************************************************************************/
/* structure to store one database object */
/* an object consists of: */
/* name of the object */
/* the name of the associated database table(s) */
/* a list of field names */
/* a where clause */
/* The function mw_connect loads the object definitions from a file */
/* into a list of this structure. */
/* The object definition in this structure is used to build the SQL-statements*/
/* which are send to the oracle data base. See mw_query, mw_insert, mw_update */
/* and mw_delete for details. */
/****************************************************************************/

struct dtb_obj_typ{
 char *name;
 char *dtb_table;
 char *sli;
 char *where;
 struct dtb_obj_typ *next_obj;
 };

/****************************************************************************/
/* The Variable objects is used to store the definition of several objects. */
/* When the application connects to Oracle, the object definition file is */
/* read. */
/* The mw_disconnect function frees the memory ocupied by the objects. */
/****************************************************************************/

struct dtb_obj_typ *objects;

struct dtb_obj_typ *actuel_object;

/****************************************************************************/
/* The result_string stores the results of the database operations */
/* The first character determins if the operation was sucessfully executed(s) */
/* or if the operation faild(f). The rest of the string contains the result */
/* of the operation (number of processed rows, rows, filenames). */
/****************************************************************************/

char result_string[VERY_LONG_STRING];
int result_string_len;

char help_no_blank[VERY_LONG_STRING];

/****************************************************************************/
/* The ora_state = 0: not connected to oracle. */
/* = 1: connected to oracle but no open coursor */
/****************************************************************************/

int ora_state = 0;

/****************************************************************************/
/* The function find_actuel_object searches for the object o_name in the */
/* list objects. If the object o_name is not found, the result is NULL. */
```

```
/**************************************************************************/

struct dtb_obj_typ *find_actuel_object(o_name)
char *o_name;
{
 struct dtb_obj_typ *help_obj;

 help_obj = objects;

 while((help_obj != NULL) && (strcmp(help_obj->name, o_name)))
 help_obj = help_obj->next_obj;

 return(help_obj);
 }

/**************************************************************************/
/* The function append_where is used to construct a valid SQL where clause */
/* using where_string and acutel_object->where. */
/**************************************************************************/

void append_where(where_string)
char *where_string;
{
 if ((actuel_object->where[0] != '\0') ||
 (where_string[0] != '\0'))
 {
 sprintf(&sql_statement[strlen(sql_statement)]," where ");
 if ((actuel_object->where[0] != '\0') &&
 (where_string[0] != '\0'))
 sprintf(&sql_statement[strlen(sql_statement)],"(%s) and (%s)",
 actuel_object->where, where_string);
 else
 {
 if (actuel_object->where[0] != '\0')
 sprintf(&sql_statement[strlen(sql_statement)],"%s",
 actuel_object->where);
 else
 sprintf(&sql_statement[strlen(sql_statement)],"%s",
 where_string);
 }
 }
 }

/**************************************************************************/
/* The function oracle_connect connects to Oracle using the username u_name */
/* and the password p_word. */
/**************************************************************************/

int oracle_connect(u_name, p_word)
char *u_name;
char *p_word;
 {
```

```
/* SQL stmt #6
EXEC SQL BEGIN DECLARE SECTION;
*/
 char username[20];
 char password[20];
/* SQL stmt #7
 EXEC SQL VAR username IS STRING(20);
 EXEC SQL VAR password IS STRING(20);
EXEC SQL END DECLARE SECTION;
*/

 strcpy(username, u_name);
 strcpy(password, p_word);

/* SQL stmt #10
 EXEC SQL WHENEVER SQLERROR GOTO connect_error;
*/

/* SQL stmt #11
 EXEC SQL CONNECT :username IDENTIFIED BY :password;
*/
 {
 sqlstm.iters = (unsigned short)10;
 sqlstm.offset = (unsigned short)2;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)username;
 sqlstm.sqhstl[0] = (unsigned long)20;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqhstv[1] = (unsigned char *)password;
 sqlstm.sqhstl[1] = (unsigned long)20;
 sqlstm.sqindv[1] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode < 0) goto connect_error;
 }
#ifndef NOPRINT
 fprintf(stderr,"\nConnected to ORACLE as user %s.\n", username);
#endif
 return 0;

connect_error:
 fprintf(stderr,"Cannot connect to ORACLE as user %s\n", username);
 return -1;
 }

/*****************************************************************************/
/* alloc_descriptors allocates memory for bind_dp and select_dp, */
/* which are needed to execut SQL-statements. */
/*****************************************************************************/

int alloc_descriptors( size, max_vname_len, max_iname_len)
int size, max_vname_len, max_iname_len;
 {
```

```
int i;

/*
* The first sqlald parameter determines the maximum number of
* array elements in each variable in the descriptor. In
* other words, it determines the maximum number of bind
* variables or select-list items in the SQL statement.
*
* The second parameter determines the maximum length of
* strings used to hold the names of select-list items
* or placeholders. The maximum length of column
* names in ORACLE is 30, but you can allocate more or less
* as needed.
*
* The third parameter determines the maximum length of
* strings used to hold the names of any indicator
* variables. To follow ORACLE standards, the maximum
* length of these should be 30. But, you can allocate
* more or less as needed.
*/

/* bind variables are not used here */
if ((bind_dp =
sqlald(1, 0, 0)) == (SQLDA *) NULL)
{
fprintf(stderr,
"Cannot allocate memory for bind descriptor.");
return -1; /* Have to exit in this case. */
}
bind_dp->N = 1;

if ((select_dp =
sqlald (size, max_vname_len, 0)) == (SQLDA *) NULL)
{
fprintf(stderr,
"Cannot allocate memory for select descriptor.");
return -1;
}
select_dp->N = MAX_ITEMS;


/* Allocate the select indicator variable pointers. */
for (i = 0; i < MAX_ITEMS; i++)
select_dp->I[i] = (short *) malloc(sizeof(short *));

/* Allocate the select variable pointers. */
/* realloc() will be used to change the size. */
for (i = 0; i < MAX_ITEMS; i++)
select_dp->V[i] = (char *) malloc(sizeof(char));


return 0;
}


/***************************************************************************/
/* free the memory occupied by the object definitions. This function is */
/* called, where the connection to oracle is closed. */
```

```
/******************************************************************************/

void free_objects()
{
 struct dtb_obj_typ *help_obj, *help_obj2;

 help_obj = objects;
 help_obj2 = objects;

 /* if help_obj == NULL then there is nothing to do! */
 if(help_obj != NULL)
 {
 /* search for the objext at the end of the list */
 /* help_obj2 points to the second but last object, or equals to help_obj */
 /* when there is only one object left */
 while(help_obj->next_obj != NULL)
 {
 help_obj2 = help_obj;
 help_obj = help_obj->next_obj;
 }

 while(help_obj != help_obj2)
 {
 /* there are more then one object left */

 /* free the memory for the object at the end of the list */
 free(help_obj->name);
 free(help_obj->dtb_table);
 free(help_obj->sli);
 free(help_obj->where);
 free(help_obj);

 /* terminate the resulting object list with NULL */
 help_obj2->next_obj = NULL;

 /* start again at the head of the list */
 help_obj = objects;
 help_obj2 = objects;

 /* search for the object at the end of the list */
 /* help_obj2 points to the second but last object, or equals to*/
 /* help_obj when there is only one object left */
 while(help_obj->next_obj != NULL)
 {
 help_obj2 = help_obj;
 help_obj = help_obj->next_obj;
 }
 } /* while(help_obj != help_obj2) */

 /* free the memory for the last object left in the list */
 free(help_obj->name);
 free(help_obj->dtb_table);
 free(help_obj->sli);
 free(help_obj->where);
 free(help_obj);

 objects = NULL;
 } /* if(help_obj != NULL) ... */
 }

/****************************************************************************/
/* The function mw_connect connects to oracle using the function */
/* oracle_connect. In addition the definitions of database objects is loaded */
/* from the file o_file. */
/****************************************************************************/

void mw_connect(u_name, p_word, o_file)
char *u_name;
char *p_word;
char *o_file;
{ FILE *obj_file;
 char file_line[VERY_LONG_STRING];
 char line[FILE_LINE_LEN]; /* to process longer lines from the */
 /* objects file */
 int line_length = FILE_LINE_LEN; /* ajust the given constants */
 struct dtb_obj_typ *help_obj, *help_obj2;

 if (ora_state != 0)
 {
#ifndef NOPRINT
 fprintf(stderr,”\n already connected to oracle”);
#endif
 strcpy(result_string, “s”); /* command faild but oracle connection should*/
 /* be ok */
 return;
 }

 /* allocate memory for the pointer objects */
 if((objects = (struct dtb_obj_typ *)
 malloc(sizeof(struct dtb_obj_typ))) == NULL)
 fprintf(stderr,”\nno memory allocated (mw_connect, objects)”);

 help_obj = objects;

 /* open file to read the object definitions */
 if ((obj_file = fopen(o_file, “r”)) == NULL)
 {
 fprintf(stderr,”\nCan’t open file ‘%s’.”,o_file);
 fprintf(stderr,”\nNot connected to Oracle.”);
 free(objects);
 strcpy(result_string,”f”);
 return;
 }

 /* read object definitions from the opend file */
 while (file_reader(line,obj_file,file_line))
 {
 if ((help_obj->name = (char *)malloc(strlen(line))) == NULL)
 fprintf(stderr,”\nno memory allocated (mw_connect, name)”);
 strcpy(help_obj->name, line);

 if (file_reader(line,obj_file,file_line))
 {
 if ((help_obj->dtb_table =(char *)malloc(strlen(line))) == NULL)
 fprintf(stderr,”\nno memory allocated (mw_connect, dtb_table)”);
strcpy(help_obj->dtb_table, line);
}
else
{
fprintf(stderr,"\nno valid object definition(table), stop and check\n");
strcpy(result_string,"f");
return;
}

if (file_reader(line,obj_file,file_line))
{
if ((help_obj->sli = (char *)malloc(strlen(line))) == NULL)
fprintf(stderr,"\nno memory allocated (mw_connect, sli)");
strcpy(help_obj->sli, line);
}
else
{
fprintf(stderr,"\nno valid object definition (sli), stop and check\n");
strcpy(result_string,"f");
return;
}

if (file_reader(line,obj_file,file_line))
{
if ((help_obj->where = (char *)malloc(strlen(line))) == NULL)
fprintf(stderr,"\nno memory allocated (mw_connect, where)");
strcpy(help_obj->where, line);
}
else
{
fprintf(stderr,"\nno valid object definition(where), stop and check\n");
strcpy(result_string,"f");
return;
}

/* prepare pointer for the next object */
if ((help_obj->next_obj = (struct dtb_obj_typ *)
malloc(sizeof(struct dtb_obj_typ))) == NULL)
fprintf(stderr,"\nno memory allocated (mw_connect, next_obj)");
help_obj2 = help_obj;
help_obj = help_obj->next_obj;
}

/* free unnecessary allocated memory */
free(help_obj);
help_obj2->next_obj = NULL;

/* close file */
fclose(obj_file);

if (oracle_connect(u_name, p_word) != 0)
{ strcpy(result_string,"f");
return;}

if (alloc_descriptors(MAX_ITEMS, MAX_VNAME_LEN, 1) != 0)
{ strcpy(result_string,"f");
return;}
#ifndef NOPRINT
```

```
 fprintf(stderr,"\nmw_connect sucessfully executed ");
#endif
 ora_state = 1;
 strcpy(result_string,"s");
 }

/****************************************************************************/
/* The mw_commit function commits all changes in the oracle database, made */
/* since the last commit of rollback. */
/****************************************************************************/

void mw_commit()
{ if (ora_state == 0)
 { fprintf(stderr,"\n not connected to oracle, no commit possible");
 strcpy(result_string,"f");
 }
 else
 {
/* SQL stmt #12
EXEC SQL WHENEVER SQLERROR CONTINUE;
 EXEC SQL COMMIT WORK;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)25;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 strcpy(result_string,"s");
 ora_state = 1;
 }
}

/****************************************************************************/
/* The mw_rollback function discarges all changes in the oracle database, */
/* made since the last commit of rollback. */
/****************************************************************************/

void mw_rollback()
{ if (ora_state == 0)
 {
 fprintf(stderr,"\n not connected to oracle, no rollback possible");
 strcpy(result_string,"f");
 }
 else
 {
/* SQL stmt #14
EXEC SQL WHENEVER SQLERROR CONTINUE;
 EXEC SQL ROLLBACK WORK;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)36;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 strcpy(result_string,"s");
 ora_state = 1;
 }
 }


/*******************************************************************************/
/* The function mw_query executes a select statement defined by the object */
/* o_name and the additional where clause in where_string. The SQL-statemant */
/* has the following form: */
/* SELECT actuel_object->sli FROM actuel_object->dtb_table */
/* WHERE actuel_object->where AND where_string */
/* If the string result_file_name is not empty, the result of the select */
/* statement is stored in a file of that name. Otherwise, the result is */
/* stored in the variable result_string. */
/*******************************************************************************/

void mw_query(o_name, where_string, result_file_name)
char *o_name;
char *where_string;
char *result_file_name;
{
 FILE *result_file;
 int i;
 int null_ok, precision, scale;
 char help_string[SMAL_STRING];

 /* Open result file if a name was given. */
 if(result_file_name[0])
 if((result_file = fopen(result_file_name, “w”)) == NULL)
 printf(“\nError, open %s”,result_file_name);
 result_string_len = 0;

 if (ora_state == 0)
 {
 fprintf(stderr,"\n not connected to oracle, query not executed!");
 strcpy(result_string, “f”);
 if(result_file_name[0])
 fclose(result_file);
 return;
 }

 actuel_object = find_actuel_object(o_name);

 if (actuel_object == NULL)
 {
 fprintf(stderr,"\n object (%s) not known",o_name);
 strcpy(result_string, “f”);
 if(result_file_name[0])
 fclose(result_file);
 return;
 }

 /* build select statement */
 sprintf(sql_statement,"select %s from %s ",
 actuel_object->sli, actuel_object->dtb_table);
 append_where(where_string);
#ifndef NOSTATEMENTS
 fprintf(stderr,"\nexecuted select statement: \n%s;\n",sql_statement);
#endif
 /* Prepare the statement and declare a cursor. */
/* SQL stmt #16
 EXEC SQL WHENEVER SQLERROR GOTO sql_error;
*/

 parse_flag = 1; /* Set a flag for sql_error. */
/* SQL stmt #17
 EXEC SQL PREPARE S FROM :sql_statement;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)47;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
 sqlstm.sqhstl[0] = (unsigned long)1024;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode < 0) goto sql_error;
}
 parse_flag = 0; /* Unset the flag. */

/* SQL stmt #18
 EXEC SQL DECLARE C CURSOR FOR S;
*/

 /* Set the bind variables for any placeholders in the
 SQL statement. */

 /* Describe any bind variables (input host variables) */

 bind_dp->N = 1; /*Initialize count of array elements. no variables used*/
/* SQL stmt #19
 EXEC SQL DESCRIBE BIND VARIABLES FOR S INTO bind_dp;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)62;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)bind_dp;
 sqlstm.sqhstl[0] = (unsigned long)0;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode < 0) goto sql_error;
}
```

```
/* If F is negative, there were more bind variables
than originally allocated by sqlald(). */
if (bind_dp->F != 0)
{
fprintf(stderr,“\n bind variables not allowed!”);
strcpy(result_string, “f”);
if(result_file_name[0])
fclose(result_file);
return;
}

/* Set the maximum number of array elements in the
descriptor to the number found. */
bind_dp->N = bind_dp->F;


/* Open the cursor and execute the statement.*/

/* SQL stmt #20
EXEC SQL OPEN C USING DESCRIPTOR bind_dp;
*/
{
sqlstm.stmt = “”;
sqlstm.iters = (unsigned short)1;
sqlstm.offset = (unsigned short)77;
sqlstm.cud = sqlcud0;
sqlstm.sqlest = (unsigned char *)&sqlca;
sqlstm.sqlety = (unsigned short)0;
sqlstm.sqhstv[0] = (unsigned char *)bind_dp;
sqlstm.sqhstl[0] = (unsigned long)0;
sqlstm.sqindv[0] = (short *)0;
sqlstm.sqphsv = sqlstm.sqhstv;
sqlstm.sqphsl = sqlstm.sqhstl;
sqlstm.sqpind = sqlstm.sqindv;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) goto sql_error;
}


/* Describe the
select-list items. The DESCRIBE function returns
their names, datatypes, lengths (including precision
and scale), and NULL/NOT NULL statuses. */

select_dp->N = MAX_ITEMS;

/* SQL stmt #21
EXEC SQL DESCRIBE SELECT LIST FOR S INTO select_dp;
*/
{
sqlstm.iters = (unsigned short)1;
sqlstm.offset = (unsigned short)92;
sqlstm.cud = sqlcud0;
sqlstm.sqlest = (unsigned char *)&sqlca;
sqlstm.sqlety = (unsigned short)0;
sqlstm.sqhstv[0] = (unsigned char *)select_dp;
sqlstm.sqhstl[0] = (unsigned long)0;
sqlstm.sqindv[0] = (short *)0;
sqlstm.sqphsv = sqlstm.sqhstv;
sqlstm.sqphsl = sqlstm.sqhstl;
sqlstm.sqpind = sqlstm.sqindv;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) goto sql_error;
}

/* If F is negative, there were more select-list
items than originally allocated by sqlald(). */
if (select_dp->F < 0)
{
fprintf(stderr,"\nToo many select-list items (%d), maximum is %d\n",
-(select_dp->F), MAX_ITEMS);
strcpy(result_string, "f");
if(result_file_name[0])
fclose(result_file);
return;
}

/* Set the maximum number of array elements in the
descriptor to the number found. */
select_dp->N = select_dp->F;

/* Allocate storage for each select-list item.

sqlprc() is used to extract precision and scale
from the length (select_dp->L[i]).

sqlnul() is used to reset the high-order bit of
the datatype and to check whether the column
is NOT NULL.

CHAR datatypes have length, but zero precision and
scale. The length is defined at CREATE time.

NUMBER datatypes have precision and scale only if
defined at CREATE time. If the column
definition was just NUMBER, the precision
and scale are zero, and you must allocate
the required maximum length.

DATE datatypes return a length of 7 if the default
format is used. This should be increased to
9 to store the actual date character string.
If you use the TO_CHAR funcRT\ tion, the maximum
length could be 75, but will probably be less
(you can see the effects of this in SQL*Plus).

ROWID datatype always returns a fixed length of 18 if
coerced to CHAR.

LONG and
LONG RAW datatypes return a length of 0 (zero),
so you need to set a maximum. In this example,
it is 240 characters.

*/
```

```
for (i = 0; i < select_dp->F; i++)
{
/* Turn off high-order bit of datatype (in this example,
it does not matter if the column is NOT NULL). */
sqlnul (&(select_dp->T[i]), &(select_dp->T[i]), &null_ok);

switch (select_dp->T[i])
{
case 1 : /* CHAR datatype: no change in length
needed, except possibly for TO_CHAR
conversions (not handled here). */
break;
case 2 : /* NUMBER datatype: use sqlprc() to
extract precision and scale. */
sqlprc (&(select_dp->L[i]), &precision, &scale);
/* Allow for maximum size of NUMBER. */
if (precision == 0) precision = 40;
/* Also allow for decimal point and
possible sign. */
select_dp->L[i] = precision + 2;
/* Allow for a negative scale. */
if (scale < 0)
select_dp->L[i] += -scale;
break;

case 8 : /* LONG datatype */
select_dp->L[i] = 240;
break;

case 11 : /* ROWID datatype */
select_dp->L[i] = 18;
break;

case 12 : /* DATE datatype */
select_dp->L[i] = 9;
break;

case 23 : /* RAW datatype */
break;

case 24 : /* LONG RAW datatype */
select_dp->L[i] = 240;
break;
}


/* Allocate space for the select-list data values. */
select_dp->V[i] = (char *) realloc(select_dp->V[i],
(size_t) select_dp->L[i]);

/* Print column headings, right-justifying number
column headings. */
#ifndef NOSTATEMENTS
if (select_dp->T[i] == 2)
fprintf(stderr,"%.*s “, select_dp->L[i], select_dp->S[i]);
else
fprintf(stderr,"%-.*s “, select_dp->L[i], select_dp->S[i]);
```

```
#endif

 /* Coerce ALL datatypes except for LONG RAW to
 character. */
 if (select_dp->T[i] != 24)
 select_dp->T[i] = 1;
 }
#ifndef NOSTATEMENTS
 fprintf(stderr,"\n\n");
#endif

 /* FETCH all rows selected and print the column values. */
/* SQL stmt #22
 EXEC SQL WHENEVER NOT FOUND GOTO end_select_found;
*/

 strcpy(result_string, “s”);

 while(1)
 {
/* SQL stmt #23
 EXEC SQL FETCH C USING DESCRIPTOR select_dp;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)107;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)select_dp;
 sqlstm.sqhstl[0] = (unsigned long)0;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode == 1403) goto end_select_found;
 if (sqlca.sqlcode < 0) goto sql_error;
}

 /* Since each variable returned has been coerced to a
 character string, very little processing is required
 here. This routine just prints out the values on
 the terminal. */
#ifndef NOSTATEMENTS
 for (i = 0; i < select_dp->F; i++)
 {
 if (*select_dp->I[i] < 0)
 fprintf(stderr,"%-*c “,(int)select_dp->L[i], ‘ ’);
 else
 fprintf(stderr,"%-*.*s “, (int)select_dp->L[i],
 (int)select_dp->L[i], select_dp->V[i]);
 }
 fprintf(stderr,"\n");
#endif
 if(result_file_name[0])
 { /* print result to a file */
 for (i = 0; i < select_dp->F; i++)
{
strcpy(help_no_blank,select_dp->V[i]);
help_no_blank[select_dp->L[i]] = '\0';
if (*select_dp->I[i] < 0)
strcpy(help_string,"\n\"\"");
else
sprintf(help_string,
"\n\"%s\"", no_blanks(help_no_blank));
fputs(help_string, result_file);
}
}
else
{ /* print result to the result_string */
for (i = 0; i < select_dp->F; i++)
{
strcpy(help_no_blank,select_dp->V[i]);
help_no_blank[select_dp->L[i]] = '\0';
if ((*select_dp->I[i] < 0) &&
(strlen(result_string)+4<VERY_LONG_STRING))
strcpy(&result_string[strlen(result_string)],":0:");
else
{
sprintf(help_string,"%s", no_blanks(help_no_blank));
if(strlen(result_string)+3+strlen(help_string)<VERY_LONG_STRING)
sprintf(&result_string[strlen(result_string)],
":%d:", strlen(help_string));
}
if ((!(*select_dp->I[i] < 0)) &&
(strlen(result_string)+1<VERY_LONG_STRING))
sprintf(&result_string[strlen(result_string)],
"%s", no_blanks(help_no_blank));
}
}
}

end_select_found:

/* Tell user how many rows processed. */
#ifndef NOSTATEMENTS
fprintf(stderr,"\n\n%d row%c processed.\n", sqlca.sqlerrd[2],
sqlca.sqlerrd[2] == 1 ? '\0' : 's');
#endif
ora_state = 1;
if(result_file_name[0])
fclose(result_file);
return;

sql_error:

/* ORACLE error handler */
fprintf(stderr,"\n\n%.70s\n",sqlca.sqlerrm.sqlerrmc);
if (parse_flag)
{
fprintf(stderr,"\nexecuted select statement: \n%s;\n",sql_statement);
fprintf(stderr,
"The parse error was at character offset %d in the SQL statement.\n",
sqlca.sqlerrd[4]);
}
```

```
/* SQL stmt #24
 EXEC SQL WHENEVER SQLERROR CONTINUE;
 EXEC SQL ROLLBACK WORK;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)122;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 ora_state = 1;
 strcpy(result_string,"f");
 if(result_file_name[0])
 fclose(result_file);
 return;
 }


/*************************************************************************/
/* The function mw_disconnect closes the connection to Oracle, after */
/* all changes, made since the last commit, are rolledback. */
/* The memory occupied by the data base object definitions is freed. */
/*************************************************************************/

void mw_disconnect()
{ int i;

 if (ora_state == 0)
 {
#ifndef NOPRINT
 fprintf(stderr,"\n not connected to oracle, no disconnect possible");
#endif
 strcpy(result_string,"f");
 return;
 }

 free_objects();
 /* When done, free the memory allocated for
 pointers in the bind and select descriptors. */
 for (i = 0; i < MAX_ITEMS; i++)
 {
 if (select_dp->V[i] != (char *) NULL)
 free(select_dp->V[i]);
 free(select_dp->I[i]); /* MAX_ITEMS were allocated. */
 }

 /* Free space used by the descriptors themselves. */
 sqlclu(bind_dp);
 sqlclu(select_dp);

/* SQL stmt #26
 EXEC SQL WHENEVER SQLERROR CONTINUE;
*/
 /* Close the cursor. */
/* SQL stmt #27
```

```
 EXEC SQL CLOSE C;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)133;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
/* SQL stmt #28
 EXEC SQL ROLLBACK WORK RELEASE;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)144;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 ora_state = 0;
 strcpy(result_string,”s”);
}

/***************************************************************************/
/* The function mw_insert executes an insert statement, defined by the given */
/* object name o_name and the data_string. The used SQL-statement has the */
/* following format: */
/* INSERT INTO actuel_object->dtb_table (actuel_object->sli) */
/* VALUES (data_string) */
/* This function is restricted to objects, that refere to only one databae */
/* table and where all fields with NOT NULL values are in the */
/* field list (sli). */
/* The data_string must contain the new values for all fields in the field */
/* list, in the same order. */
/***************************************************************************/

void mw_insert(o_name, data_string)
char *o_name;
char *data_string;
{ int i;

 if (ora_state == 0)
 {
 fprintf(stderr,”\n not connected to oracle, insert not executed!”);
 strcpy(result_string, “f”);
 return;
 }

 actuel_object = find_actuel_object(o_name);

 if (actuel_object == NULL)
 {
 fprintf(stderr,”\n object not known”);
 strcpy(result_string, “f”);
 return;
 }
 /* build insert statement */
 sprintf(sql_statement,"insert into %s (%s) values (%s)",
 actuel_object->dtb_table, actuel_object->sli,
 data_string);
#ifndef NOSTATEMENTS
 fprintf(stderr,"\nexecuted insert statement: \n%s;\n",sql_statement);
#endif
 /* Prepare the statement and declare a cursor. */
/* SQL stmt #29
 EXEC SQL WHENEVER SQLERROR GOTO sql_error;
*/

 parse_flag = 1; /* Set a flag for sql_error. */
/* SQL stmt #30
 EXEC SQL PREPARE S FROM :sql_statement;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)155;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
 sqlstm.sqhstl[0] = (unsigned long)1024;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode < 0) goto sql_error;
}
 parse_flag = 0; /* Unset the flag. */

/* SQL stmt #31
 EXEC SQL EXECUTE IMMEDIATE :sql_statement;
*/
{
 sqlstm.stmt = "";
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)170;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
 sqlstm.sqhstl[0] = (unsigned long)1024;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode == 1403) goto end_select_found;
 if (sqlca.sqlcode < 0) goto sql_error;
}

 /* Tell user how many rows processed. */
#ifndef NOSTATEMENTS
 fprintf(stderr,"\n\n%d row%c processed.\n", sqlca.sqlerrd[2],
 sqlca.sqlerrd[2] == 1 ? '\0' : 's');
#endif
 ora_state = 1;
 sprintf(result_string,"s:%d",sqlca.sqlerrd[2]);
 return;

 sql_error:

 /* ORACLE error handler */
 fprintf(stderr,"\n\n%.70s\n",sqlca.sqlerrm.sqlerrmc);
 if (parse_flag)
 {
 fprintf(stderr,"\nexecuted insert statement: \n%s;\n",sql_statement);
 fprintf(stderr,
 "The parse error was at character offset %d in the SQL statement.\n",
 sqlca.sqlerrd[4]);
 }

/* SQL stmt #32
 EXEC SQL WHENEVER SQLERROR CONTINUE;
 EXEC SQL ROLLBACK WORK;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)185;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 ora_state = 1;
 strcpy(result_string,"f");
 return;

 end_select_found:
 ora_state = 1;
 strcpy(result_string,"f");
 return;

}

/*****************************************************************************/
/* The function mw_update executs an update statement, defined by the object */
/* o_name, the additional where clause where_string. The fields to update */
/* and the new values are in the string data_string. Updates are only allowed */
/* for objects, refering to only one database table. */
/* The used SQL-statement has the following format: */
/* UPDATE actuel_object->dtb_table SET data_string */
/* WHERE actuel_object->where AND where_string */
/*****************************************************************************/

void mw_update(o_name, data_string, where_string)
char *o_name;
char *data_string;
char *where_string;
{ int i;

 if (ora_state == 0)
```

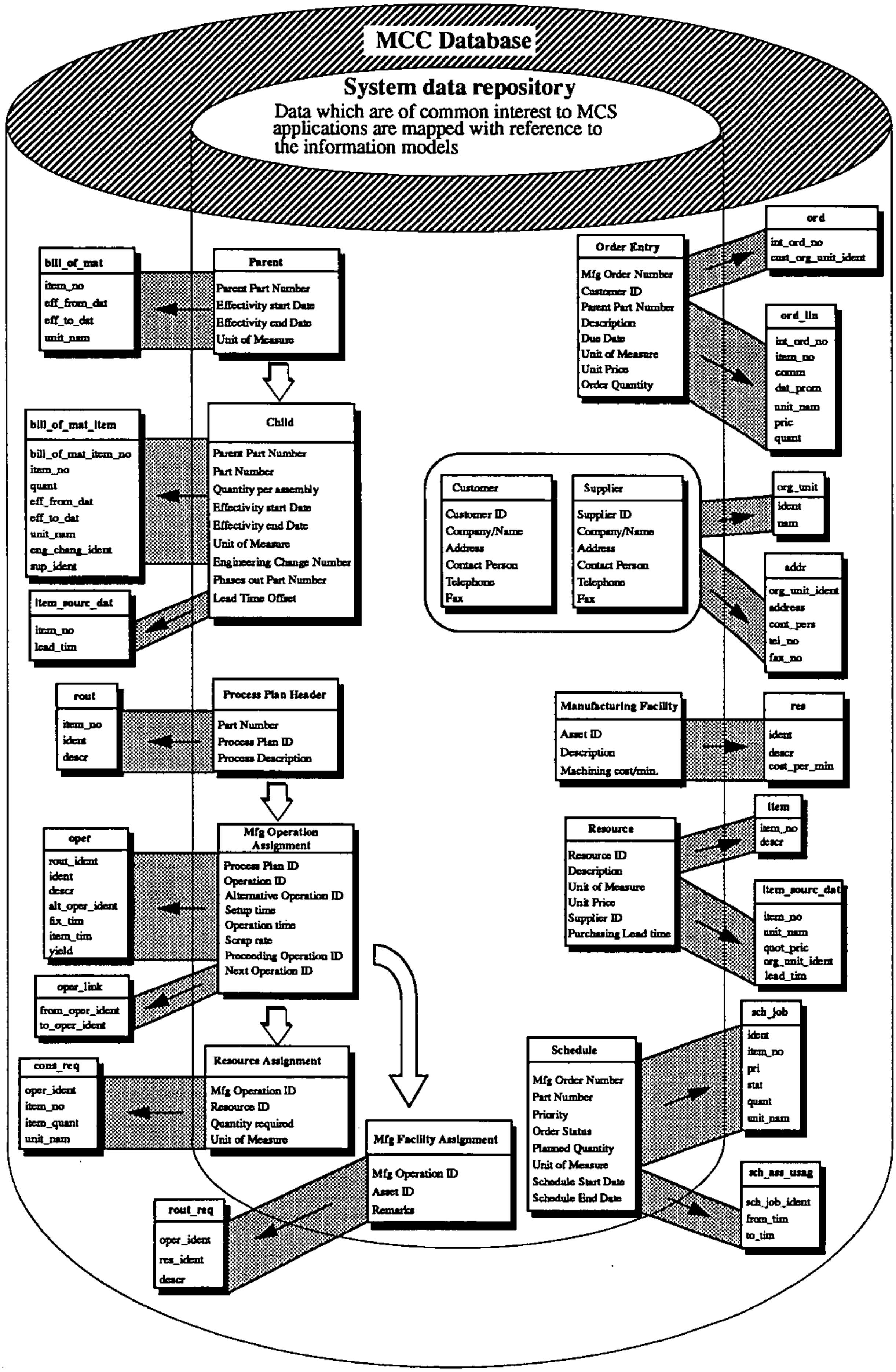
```
{
fprintf(stderr,”\n not connected to oracle, update not executed!”);

strcpy(result_string, “f”);
return;
}

actuel_object = find_actuel_object(o_name);

if (actuel_object == NULL)
{
fprintf(stderr,”\n object not known”);
strcpy(result_string, “f”);
return;
}

/* build update statement */
sprintf(sql_statement,”update %s set %s “,
actuel_object->dtb_table, data_string);
append_where(where_string);
#ifndef NOSTATEMENTS
fprintf(stderr,”\nexecuted update statement: \n%s;\n”,sql_statement);
#endif
/* Prepare the statement and declare a cursor. */
/* SQL stmt #34
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
*/

parse_flag = 1; /* Set a flag for sql_error. */
/* SQL stmt #35
EXEC SQL PREPARE S FROM :sql_statement;
*/
{
sqlstm.iters = (unsigned short)1;
sqlstm.offset = (unsigned short)196;
sqlstm.cud = sqlcud0;
sqlstm.sqlest = (unsigned char *)&sqlca;
sqlstm.sqlety = (unsigned short)0;
sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
sqlstm.sqhstl[0] = (unsigned long)1024;
sqlstm.sqindv[0] = (short *)0;
sqlstm.sqphsv = sqlstm.sqhstv;
sqlstm.sqphsl = sqlstm.sqhstl;
sqlstm.sqpind = sqlstm.sqindv;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) goto sql_error;
}
parse_flag = 0; /* Unset the flag. */

/* SQL stmt #36
EXEC SQL EXECUTE IMMEDIATE :sql_statement;
*/
{
sqlstm.stmt = “”;
sqlstm.iters = (unsigned short)1;
sqlstm.offset = (unsigned short)211;
sqlstm.cud = sqlcud0;
sqlstm.sqlest = (unsigned char *)&sqlca;
```

```
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
 sqlstm.sqhstl[0] = (unsigned long)1024;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode == 1403) goto end_select_found;
 if (sqlca.sqlcode < 0) goto sql_error;
 }

 /* Tell user how many rows processed. */
#ifndef NOSTATEMENTS
 fprintf(stderr,"\n\n%d row%c processed.\n", sqlca.sqlerrd[2],
 sqlca.sqlerrd[2] == 1 ? '\0' : 's');
#endif
 ora_state = 1;
 sprintf(result_string,"s:%d",sqlca.sqlerrd[2]);
 return;

 sql_error:

 /* ORACLE error handler */
 fprintf(stderr,"\n\n%.70s\n",sqlca.sqlerrm.sqlerrmc);
 if (parse_flag)
 {
 fprintf(stderr,"\nexecuted update statement: \n%s;\n",sql_statement);
 fprintf(stderr,
 "The parse error was at character offset %d in the SQL statement.\n",
 sqlca.sqlerrd[4]);
 }

/* SQL stmt #37
 EXEC SQL WHENEVER SQLERROR CONTINUE;
 EXEC SQL ROLLBACK WORK;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)226;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 ora_state = 1;
 strcpy(result_string,"f");
 return;

 end_select_found:
 ora_state = 1;
 sprintf(result_string,"s:%d",sqlca.sqlerrd[2]);
 return;
}

/*************************************************************************/
/* The function mw_delete executes a delete statement, defined by the object */
/* o_name and the additional where clause in where_string. It is restricted */
/* to objects, refering to only one database table. */
/* The used SQL-statement has the following format: */
/* DELETE FROM actuel_object->dtb_table */
/* WHERE actuel_object->where AND where_string */
/*****************************************************************************/

void mw_delete(o_name, where_string)
char *o_name;
char *where_string;
{ int i;

 if (ora_state == 0)
 {
 fprintf(stderr,"\n not connected to oracle, delete not executed!");

 strcpy(result_string, "f");
 return;
 }

 actuel_object = find_actuel_object(o_name);

 if (actuel_object == NULL)
 {
 fprintf(stderr,"\n object not known");
 strcpy(result_string, "f");
 return;
 }

 /* build delete statement */
 sprintf(sql_statement,"delete from %s ",
 actuel_object->dtb_table);
 append_where(where_string);
#ifndef NOSTATEMENTS
 fprintf(stderr,"\nexecuted delete statement: \n%s;\n",sql_statement);
#endif
 /* Prepare the statement and declare a cursor. */
/* SQL stmt #39
 EXEC SQL WHENEVER SQLERROR GOTO sql_error;
*/

 parse_flag = 1; /* Set a flag for sql_error. */
/* SQL stmt #40
 EXEC SQL PREPARE S FROM :sql_statement;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)237;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
 sqlstm.sqhstl[0] = (unsigned long)1024;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode < 0) goto sql_error;
}
 parse_flag = 0; /* Unset the flag. */

/* SQL stmt #41
 EXEC SQL EXECUTE IMMEDIATE :sql_statement;
*/
{
 sqlstm.stmt = ‘“‘;
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)252;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlstm.sqhstv[0] = (unsigned char *)sql_statement;
 sqlstm.sqhstl[0] = (unsigned long)1024;
 sqlstm.sqindv[0] = (short *)0;
 sqlstm.sqphsv = sqlstm.sqhstv;
 sqlstm.sqphsl = sqlstm.sqhstl;
 sqlstm.sqpind = sqlstm.sqindv;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
 if (sqlca.sqlcode == 1403) goto end_select_found;
 if (sqlca.sqlcode < 0) goto sql_error;
}

 /* Tell user how many rows processed. */
#ifndef NOSTATEMENTS
 fprintf(stderr,"\n\n%d row%c processed.\n”, sqlca.sqlerrd[2],
 sqlca.sqlerrd[2] == 1 ? '\0’ : ‘s’);
#endif
 ora_state = 1;
 sprintf(result_string,”s:%d”,sqlca.sqlerrd[2]);
 return;

 sql_error:

 /* ORACLE error handler */
 fprintf(stderr,"\n\n%.70s\n”,sqlca.sqlerrm.sqlerrmc);
 if (parse_flag)
 {
 fprintf(stderr,"\nexecuted delete statement: \n%s;\n”,sql_statement);
 fprintf(stderr,
 “The parse error was at character offset %d in the SQL statement.\n”,
 sqlca.sqlerrd[4]);
 }

/* SQL stmt #42
 EXEC SQL WHENEVER SQLERROR CONTINUE;
 EXEC SQL ROLLBACK WORK;
*/
{
 sqlstm.iters = (unsigned short)1;
 sqlstm.offset = (unsigned short)267;
 sqlstm.cud = sqlcud0;
 sqlstm.sqlest = (unsigned char *)&sqlca;
 sqlstm.sqlety = (unsigned short)0;
 sqlcex(&sqlctx, &sqlstm, &sqlfpn);
}
 ora_state = 1;
```

```
strcpy(result_string,”f”);
return;

end_select_found:
ora_state = 1;
sprintf(result_string,”s:%d”,sqlca.sqlerrd[2]);
return;
}
```

APPENDIX VI

Mapping of proprietary MCC information entities to the information models in the system data repository


MCC Database
System data repository
Data which are of common interest to MCS applications are mapped with reference to the information models
bill_of_mat
item_no
eff_from_dat
eff_to_dat
unit_nam
Parent
Parent Part Number
Effectivity start Date
Effectivity end Date
Unit of Measure
bill_of_mat_item
bill_of_mat_item_no
item_no
quant
eff_from_dat
eff_to_dat
unit_nam
eng_chang_ident
sup_ident
Child
Parent Part Number
Part Number
Quantity per assembly
Effectivity start Date
Effectivity end Date
Unit of Measure
Engineering Change Number
Phases out Part Number
Lead Time Offset
item_sourc_dat
item_no
lead_tim
rout
item_no
ident
descr
Process Plan Header
Part Number
Process Plan ID
Process Description
oper
rout_ident
ident
descr
alt_oper_ident
fix_tim
item_tim
yield
Mfg Operation Assignment
Process Plan ID
Operation ID
Alternative Operation ID
Setup time
Operation time
Scrap rate
Proceeding Operation ID
Next Operation ID
oper_link
from_oper_ident
to_oper_ident
cons_req
oper_ident
item_no
item_quant
unit_nam
Resource Assignment
Mfg Operation ID
Resource ID
Quantity required
Unit of Measure
Mfg Facility Assignment
Mfg Operation ID
Asset ID
Remarks
rout_req
oper_ident
res_ident
descr
Order Entry
Mfg Order Number
Customer ID
Parent Part Number
Description
Due Date
Unit of Measure
Unit Price
Order Quantity
ord
int_ord_no
cust_org_unit_ident
ord_lin
int_ord_no
item_no
comm
dat_prom
unit_nam
pric
quant
Customer
Customer ID
Company/Name
Address
Contact Person
Telephone
Fax
Supplier
Supplier ID
Company/Name
Address
Contact Person
Telephone
Fax
org_unit
ident
nam
addr
org_unit_ident
address
cont_pers
tel_no
fax_no
Manufacturing Facility
Asset ID
Description
Machining cost/min.
res
ident
descr
cost_per_min
Resource
Resource ID
Description
Unit of Measure
Unit Price
Supplier ID
Purchasing Lead time
item
item_no
descr
item_sourc_dat
item_no
unit_nam
quot_pric
org_unit_ident
lead_tim
Schedule
Mfg Order Number
Part Number
Priority
Order Status
Planned Quantity
Unit of Measure
Schedule Start Date
Schedule End Date
sch_job
ident
item_no
pri
stat
quant
unit_nam
sch_ass_usag
sch_job_ident
from_tim
to_tim

Program listings for mapping and population of the following data from MCC to data repository:

1. Manufacturing resources and process plans (Create_process_file.c)

2. Schedule (Create_schedule.c)

3. WIP (Populate_wip.c)

4. BOM (populate_bom.c)

Create_process_file.c

```
/*************************************************************************/
/* File: create_process_file.c */
/* This program creates a file with process information, which is readable */
/* by the cell controler. Before the process information can be */
/* put in a file with this program, the file with the schedule information */
/* must be created, using the program create_schedule_file. */
/* Then this program extracts the process information for all parts, */
/* which are in that schedule. */
/*************************************************************************/

#include <stdio.h>
#include “local_incl.h”

/*************************************************************************/
/* The string result_string is used to communicate with the functions of the */
/* program access_ora, with are used to access the oracle database. */
/*************************************************************************/
extern char result_string[VERY_LONG_STRING];

char help[LONG_STRING];
char value[SMAL_STRING];
FILE *drive, *drive2;
char file_rest[FILE_LINE_LEN];

char temp_sel[MEDIUM_STRING];
char process_data_file[MEDIUM_STRING];

/*************************************************************************/
/* The function init_oracle connects to the MCC oracle database and sets */
/* values for temporal files and the file with the process data. */
/* The file with the schedule data is opened for writeing. */
/*************************************************************************/

void init_oracle()
{
 strcpy(temp_sel,”/home2/sandra/valdew/misc/temp”);

 strcpy(process_data_file,
 “/home/wayne/blzhang/iclcell/message/part_proc.data”);

 mw_connect(“mcc21”,”m”,
 “/home2/sandra/valdew/oracle_c/embed_sql/config_tables_process.txt”);

 if((drive = fopen(process_data_file,”w”)) == NULL)
 printf(“\nError, open %s.”,process_data_file);
}

/*************************************************************************/
/* The function terminate_oracle closes the process_data_file, commits all */
/* changes made in the oracle database and closes the connection to the */
/* oracle database. */
/*************************************************************************/
```

```
void terminate_oracle()
{
 fclose(drive);
 mw_commit();
 mw_disconnect();
 printf(“\n”);
}

/***********************************************************************/
/* The function write_process writes the process data to the file */
/* process_data_file, for all parts in the current schedule. */
/***********************************************************************/

void write_process()
{
 char part_no[SMAL_STRING], process_id[SMAL_STRING];
 char part_type[SMAL_STRING], operation_time[SMAL_STRING];
 char mfg_operation_id[SMAL_STRING];
 char asset_id[SMAL_STRING], asset_number[SMAL_STRING];
 int operation_number;

 /* Write header to the process_data_file. */

 sprintf(help,"Part Process Data File\n");
 fputs(help,drive);

 /* The table help_asset_number is used to generate machine numbers for */
 /* assets, related to the operations. As the cell controler only accepts */
 /* machine numbers between 1 and 4, the asset_id can't be used directly. */
 /* This information is stored permanently in the database to be able to */
 /* interpret the feed back information, produced by the cell controller, */
 /* correctly. */

 mw_delete(“help_asset_number_1”,“”);
 sprintf(help,"0, ‘0’");
 mw_insert(“help_asset_number_1”,help);

 /* Retrieve all parts, which are produced by the current schedule. */
 /* The table help_part_type is used for this purpos, because it contains */
 /* the part type for all parts. (See file create_schedule_file for */
 /* details. */

 mw_query(“help_part_type_1”,"part_type > 0 order by part_type”,temp_sel);
 if((drive2 = fopen(temp_sel,"r")) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;

 while(file_reader(part_no, drive2, file_rest))
 {
file_reader(part_type, drive2, file_rest);

/* Print the header for a process rout to the process_data_file. */
/* The number of machines needed to produce the current part is */
/* set to 4 by default. */

 sprintf(help,"--------------------------------------\n");
 fputs(help,drive);
sprintf(help,"%s\t%d\n",part_type, 4);
fputs(help,drive);

/* Get the process_plan_id for the current part from the */
/* process_plan_id table.  */

sprintf(help,"part_number='%s'",part_no);
mw_query("process_plan_id_1",help,"");
next_value(result_string,process_id,result_string);

/* Get the first operation for the current part. The first operation */
/* is identified by having no preceeding operation but having a next */
/* operation. If the part has only one operation, its process rout */
/* will not be extracted correctly. In the current version, */
/* alternative operations are not extracted.  */

sprintf(help,"o.process_plan_id=%s and o.presc_mfg_operation_id IS NULL and o.next_mfg_operation_id IS
NOT NULL", process_id);
mw_query("mfg_operation_ass_1",help,"");

operation_number = 0;

while(next_value(result_string,mfg_operation_id,result_string))
{
 next_value(result_string,operation_time,result_string);
 next_value(result_string, asset_id, result_string);
 operation_number++;

 /* Get the machine number for the current asset. */

 sprintf(help,"asset_id = %s", asset_id);
 mw_query("help_asset_number_2",help,"");
 if(!next_value(result_string, asset_number, result_string))
 {
 /* Generate a new number for the current asset. */

 mw_query("help_asset_number_3","","");
 next_value(result_string,asset_number,result_string);
 sprintf(help,"%s, '%s'",asset_id, asset_number);
 mw_insert("help_asset_number_1",help);
 }

 /* Write the operation details to the process_data_file. */

 sprintf(help,"%d\t%s\t%s\t0\n",operation_number, asset_number,
operation_time);
 fputs(help,drive);

 /* Get the next operation. */

 sprintf(help,"o.process_plan_id=%s and o.presc_mfg_operation_id = %s", process_id, mfg_operation_id);
 mw_query("mfg_operation_ass_1",help,"");
 }

fputs("0\n",drive);
 }
 fclose(drive2);
 }
```

```
main()
{
 init_oracle();
 write_process();
 terminate_oracle();
}
```

Meta-file definition

Name of meta-file : config_tables_process.txt

Definition of database-objects: (for access_ora)

```
“help_part_type_1” “help_part_type”
“part_no, part_type” ““‘

“help_asset_number_1” “help_asset_number”
“asset_id, asset_number” ““‘

“help_asset_number_2” “help_asset_number”
“asset_number” ““‘

“help_asset_number_3” “help_asset_number”
“(max(asset_number) + 1)” ““‘

“process_plan_id_1” “process_plan_id”
“process_plan_id” ““‘

“mfg_operation_ass_1” “mfg_operation_ass o, mfg_facility_ass a”
“o.mfg_operation_id, o.operation_time, a.asset_id”
“a.mfg_operation_id = o.mfg_operation_id”
```

Create_schedule_file.c

```
/******************************************************************************/
/* File: create_schedule_file.c */
/* This program creates a file with scheduling information, which is readable */
/* by the cell controler. */
/******************************************************************************/

#include <stdio.h>
#include “local_incl.h”

/******************************************************************************/
/* The string result_string is used to communicate with the functions of the */
/* program access_ora, with are used to access the oracle database. */
/******************************************************************************/
extern char result_string[VERY_LONG_STRING];

char help[LONG_STRING];
char value[SMAL_STRING], result_rest[VERY_LONG_STRING];
FILE *drive, *drive2;
char file_rest[FILE_LINE_LEN];

char temp_sel[MEDIUM_STRING];
char schedule_data_file[MEDIUM_STRING];

/******************************************************************************/
/* The function init_oracle connects to the MCC oracle database and sets */
/* values for temporal files and the file with the schedule data. */
/* The file with the schedule data is opened for writeing. */
/******************************************************************************/

void init_oracle()
{
strcpy(temp_sel,
 “/home2/sandra/valdew/misc/temp”);

strcpy(schedule_data_file,
 “/home/wayne/blzhang/iclcell/message/schedule.data”);

mw_connect(“mcc21”,”m”,
“/home2/sandra/valdew/oracle_c/embed_sql/config_tables_schedule.txt”);

 if((drive = fopen(schedule_data_file,”w”)) == NULL)
printf(“\nError, open %s.”,schedule_data_file);
}

/******************************************************************************/
/* The function terminate_oracle writes the tail to the schedule_data_file, */
/* commits changes made in the oracle database and ends the oracle session. */
/******************************************************************************/

void terminate_oracle()
{
 sprintf(help,”\t000\n”,value);
 fputs(help,drive);
 sprintf(help,”----------------------------------------\n”);
 fputs(help,drive);
 fclose(drive);
 mw_commit();
 mw_disconnect();
 printf("\n");
 }

/****************************************************************************/
/* The function write_schedule_header writes the header to the file */
/* schedule_data_file. */
/****************************************************************************/

void write_schedule_header()
{
 sprintf(help,"\t Production Schedule\n");
 fputs(help,drive);
 sprintf(help,"------------------------------------------\n");
 fputs(help,drive);
 mw_query("date","","");
 next_value(result_string,value,result_rest);
 sprintf(help,"\t001\\t%s\n",value);
 fputs(help,drive);
 sprintf(help,"------------------------------------------\n");
 fputs(help,drive);
 }

/****************************************************************************/
/* The function generate_part_types generates a unique part type for each */
/* scheduled part. This is done because the cell controller feed back is */
/* related only to part types and not to order numbers and part numbers. */
/* To be able relate the feed back to orders and parts, the part types are */
/* stored in the database table help_part_type. A process rout has to be */
/* produced for each part type as well. The program create_process_file will */
/* do that job. */
/****************************************************************************/

void generate_part_types()
{
 char mfg_order_no[SMAL_STRING], part_no[SMAL_STRING];
 char part_type[SMAL_STRING];

 /* Initialise the table help_part_type. */
 mw_delete("help_part_type_1","");

 sprintf(help,"'0', '0', 0");
 mw_insert("help_part_type_1",help);

 /* Get all parts in the schedule. */
 mw_query("schedule_1","",temp_sel);
 if((drive2 = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
 file_rest[0] = '\0';

 /* Generate a unique part type for each order and part, starting with */
 /* the part type 1. */
 while(file_reader(mfg_order_no, drive2, file_rest))
  {
file_reader(part_no, drive2, file_rest);
mw_query("next_part_type","","");
next_value(result_string, part_type, result_rest);
sprintf(help,"%s, '%s', %s",mfg_order_no, part_no, part_type);
mw_insert("help_part_type_1", help);
 }
 fclose(drive2);
}

/****************************************************************************/
/* The function write_schedule writes the schedule data to the file */
/* schedule_data_file. */
/****************************************************************************/

void write_schedule()
{
 char mfg_order_no[SMAL_STRING], part_no[SMAL_STRING];
 char part_type[SMAL_STRING], planned_quantity[SMAL_STRING];
 int schedule_order_number = 0;

 /* Get all parts in the schedule. */
 mw_query("schedule_2","",temp_sel);
 if((drive2 = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
 file_rest[0] = '\0';

 /* Write an entry for each scheduled part to the file. */
 while(file_reader(mfg_order_no, drive2, file_rest))
 {
schedule_order_number++;
file_reader(part_no, drive2, file_rest);
file_reader(planned_quantity, drive2, file_rest);

sprintf(help,"mfg_order_no=%s and part_no='%s'",mfg_order_no, part_no);
mw_query("help_part_type_2",help,"");
next_value(result_string, part_type, result_rest);
sprintf(help,"\t00%d\t\t00%s\t\t%s\n",schedule_order_number,
part_type, planned_quantity);
fputs(help,drive);

file_reader(value, drive2, file_rest);
file_reader(value, drive2, file_rest);
 }
 fclose(drive2);
}

main()
{
 init_oracle();
 write_schedule_header();
 generate_part_types();
 write_schedule();
 terminate_oracle();
}
```

Meta-file definition

Name of meta-file : config_tables_schedule.txt

Definition of database-objects: (for access_ora)

```
“help_part_type_1” “help_part_type”
“mfg_order_no, part_no, part_type” ““

“help_part_type_2” “help_part_type”
“part_type” ““

“next_part_type” “help_part_type”
“(max(part_type) + 1)” ““

“date”“dual”
“sysdate” ““

“schedule_1” “schedule”
“mfg_order_no, part_no” ““

“schedule_2” “schedule”
“mfg_order_no, part_no, planned_quantity, priority, schedule_start_date”
“planned_quantity > 0 order by schedule_start_date, priority”
```

Populate_wip.c

```
/*****************************************************************************/
/* File: populate_wip.c */
/* This program populates the data repository wip. */
/* Any old data residing in this table is deleted. */
/* The data is read from the cell status file: */
/*‘/home/wayne/blzhang/iclcell/message/cell_status.data’. */
/*****************************************************************************/

#include <stdio.h>
#include “local_incl.h”

/*****************************************************************************/
/* The string result_string is used to communicate with the functions of the */
/* program access_ora, with are used to access the oracle database. */
/*****************************************************************************/
extern char result_string[VERY_LONG_STRING];

char wip_file[MEDIUM_STRING];
FILE *drive;

/*****************************************************************************/
/* The function init_oracle connects to the MCC oracle database and opens */
/* cell status file. */
/*****************************************************************************/

void init_oracle()
{
strcpy(wip_file,”/home/wayne/blzhang/iclcell/message/cell_status.data”);
if((drive=fopen(wip_file,”r”)) == NULL)
 printf(“\nError oper file: %s.”,wip_file);

mw_connect(“mcc21”,”m”,
 “/home2/sandra/valdew/oracle_c/embed_sql/config_tables_wip_2.txt”);
}

/*****************************************************************************/
/* The function terminate_oracle closes the cell status file, commits all */
/* changes in the database and ends the database session. */
/*****************************************************************************/

void terminate_oracle()
{
 fclose(drive);
 mw_commit();
 mw_disconnect();
 printf(“\n”);
}

/*****************************************************************************/
/* The function word_no retrieves the n’th word from a string. The words */
/* have to be seperated by blanks. */
/*****************************************************************************/

char word_no(line, number, word)
char *line;
int number;
char *word;
{
 char blank_state = 1;
 int i = 0, start, end, act_no = 1;

 word[0] = ‘\0’;
 while(line[i] == ‘ ’)i++;
 if(!line[i]) return(0);
 while(act_no < number)
 {
while((line[i] != ‘ ’) && line[i])i++;
while(line[i] == ‘ ’)i++;
 if(!line[i]) return(0);
act_no++;
 }
 start = i;
 while((line[i] != ‘ ’) && line[i])i++;
 end = i;
 strncpy(word, &line[start], end-start);
 word[end-start] = ‘\0’;
 return(1);
}

/******************************************************************************/
/* The function pop_wip reads from the cell status file and puts some of the */
/* data in the table wip.  */
/******************************************************************************/

void pop_wip()
{
 char file_string[FILE_LINE_LEN], file_string2[FILE_LINE_LEN];
 char blank_state = 0;
 int i;
 int cell, time;
 char order_no[SMAL_STRING], part_no[SMAL_STRING], type[SMAL_STRING];
 char quant[SMAL_STRING];
 char help2[SMAL_STRING], help1[SMAL_STRING];

 /* Delete any old wip data. */

 mw_delete(“wip”,“”);

 printf(“\n”);

 /* Read the first three lines. */

 for(i=0;i<3;i++) fgets(file_string, FILE_LINE_LEN, drive);

 /* Get the cell number and the actual time. */

 word_no(file_string, 3, help1);
 cell = to_number(help1);
 word_no(file_string, 5, help1);
 time = to_number(help1);
 printf(“\nCell: %d, Time: %d.”,cell, time);
/* Get the status and utilisation rate of the machines. */

 for(i=0;i<5;i++) fgets(file_string, FILE_LINE_LEN, drive);
 fgets(file_string2, FILE_LINE_LEN, drive);
 for(i=1;i<=cell;i++)
 {
word_no(file_string, i, help1);
word_no(file_string2, i, help2);
printf(“\nMachine %d, State: %s, Util: %s.”,i, help1, help2);
 }

/* Get the status and utilisation rate of the AGV. */

 word_no(file_string, 5, help1);
 help1[strlen(help1)-1] = ‘\0’;
 word_no(file_string2, 5, help2);
 help2[strlen(help2)-1] = ‘\0’;
 printf(“\nAGV State: %s, Util: %s.”, help1, help2);

/* Get the number of parts in the raw part area. */

 for(i=0;i<5;i++) fgets(file_string, FILE_LINE_LEN, drive);
 printf(“\nRaw Part Area:”);
 word_no(file_string, 1, help1);
 while(strcmp(help1, “000”))
 {
 word_no(file_string, 2, help2);
 help2[strlen(help2)-1] = ‘\0’;
printf(“\nPart Type: %s, Number: %s.”,help1, help2);
fgets(file_string, FILE_LINE_LEN, drive);
word_no(file_string, 1, help1);
 }

/* Get the number of parts in the finished part area and store that */
/* data in the wip table. */

 for(i=0;i<3;i++) fgets(file_string, FILE_LINE_LEN, drive);
 printf(“\nFinished Part Area:”);
 word_no(file_string, 1, help1);
 while(strcmp(help1, “000”))
 {
 word_no(file_string, 2, quant);
 quant[strlen(quant)-1] = ‘\0’;
sprintf(type,”%d”,to_number(help1));
printf(“\nPart Type: %s, Number: %s.”,type, quant);

sprintf(help1,”part_type = %s”,type);
mw_query(“help_part_type”,help1,””);
next_value(result_string, order_no, result_string);
next_value(result_string, part_no, result_string);
printf(“\nOrder_no: %s, Part_no: %s.”,order_no, part_no);
sprintf(help1,”%s, ‘%s’, %s”,order_no, part_no, quant);
mw_insert(“wip”, help1);

fgets(file_string, FILE_LINE_LEN, drive);
word_no(file_string, 1, help1);
 }
}
```

```
main()
{
 init_oracle();
 pop_wip();
 terminate_oracle();
}
```

Meta-file definition

Name of meta-file : config_tables_wip_2.txt

Definition of database-objects: (for access_ora)

```
"help_part_type" "help_part_type"
"mfg_order_no, part_no" ""

"wip" "wip"
"mfg_order_no, part_no, actual_quantity_prod" ""
```

Populate_bom.c

```
/* File: populate_bom.c */
/* This program populates the data repositories bom_parent and bom_child. */
/* Any old data residing in those tables is deleted. */
/* The item_no’s starting with ‘LUT...’ and Bom name ‘BOM’ are chosen. */

#include <stdio.h>
#include “local_incl.h”

/****************************************************************************/
/* The string result_string is used to communicate with the functions of the */
/* program access_ora, with are used to access the oracle database. */
/****************************************************************************/
extern char result_string[VERY_LONG_STRING];

char temp_sel[MEDIUM_STRING];
int level_number;

/****************************************************************************/
/* The function init_oracle connects to the MCC oracle database and sets */
/* values for temporal files. */
/****************************************************************************/

void init_oracle()
{
strcpy(temp_sel,
“/home2/sandra/valdew/misc/temp”);

mw_connect(“mcc21”,”m”,
“/home2/sandra/valdew/oracle_c/embed_sql/config_tables_bom.txt”);
}

/****************************************************************************/
/* The function terminate_oracle commits all changes in the oracle database */
/* and ends the database session. */
/****************************************************************************/

void terminate_oracle()
{
 mw_commit();
 mw_disconnect();
 printf(“\n”);
}

/****************************************************************************/
/* The function pop_child populates the table bom_child with data from the */
/* MCC table bill_of_mat_item. It puts the data specified by child_id in */
/* the bom_child table and calls pop_child for all children of this child. */
/****************************************************************************/

void pop_child(child_id, level)
char *child_id;
int level;
{
 char help[LONG_STRING];
 FILE *drive;
```

```
char file_rest[FILE_LINE_LEN];
char loc_file[MEDIUM_STRING];

char parent_no[SMAL_STRING], s_date[SMAL_STRING], e_date[SMAL_STRING];
char child_no[SMAL_STRING];
char unit_nam[SMAL_STRING], lead_time[SMAL_STRING];
char sub_child_id[SMAL_STRING];
char chang_no[SMAL_STRING], sup_ident[SMAL_STRING], quant[SMAL_STRING];
int childs;
char pritty_print[SMAL_STRING];
int i;

pritty_print[0]='\0';
for(i=0;i<level;i++) sprintf(pritty_print,"%s ",pritty_print);

sprintf(loc_file,"%s%d",temp_sel,level);
if(level > level_number) level_number = level;
printf("\n%schild_id: %s.",pritty_print,child_id);

/* Get the data from the bill_of_mat table. */

 sprintf(help,"b.ident = %s", child_id);
 mw_query("bom_child",help,"");

 next_value(result_string, parent_no, result_string);
 next_value(result_string, child_no, result_string);
 next_value(result_string, s_date, result_string);
 next_value(result_string, e_date, result_string);
 next_value(result_string, unit_nam, result_string);
 next_value(result_string, lead_time, result_string);
 next_value(result_string, chang_no, result_string);
 next_value(result_string, sup_ident, result_string);
 next_value(result_string, quant, result_string);

/* Set non existant values to NULL. */

 if(e_date[0]) sprintf(e_date,"'%s'",e_date); else sprintf(e_date,"NULL");
 if(sup_ident[0]) sprintf(sup_ident,"'%s'",sup_ident);
else sprintf(sup_ident,"NULL");
 if(!lead_time[0]) sprintf(lead_time,"NULL");
 if(!chang_no[0]) sprintf(chang_no,"NULL");

/* Get all children. */

 sprintf(help,"bill_of_mat_item_no='%s' and bill_of_mat_nam='BOM'",child_no);
 mw_query("bill_of_mat_item_1",help,loc_file);
 if((drive=fopen(loc_file,"r")) == NULL)
printf("\nError oper file %s.",loc_file);
 file_rest[0]='\0';
 childs = 0;

/* Call pop_child for all children (if any). */

 while(file_reader(sub_child_id, drive, file_rest))
 {
pop_child(sub_child_id, (level+1));
childs++;
 }
 fclose(drive);

 printf(“\n%s%s, %s, %s, %s, %s, %s, %s, %s, %d, %s”,pritty_print,parent_no,
child_no, s_date, e_date, unit_nam,
lead_time, chang_no, sup_ident, childs, quant);

 /* Insert row in the bom_child table. */

 sprintf(help,”‘%s’, ‘%s’, ‘%s’, %s, ‘%s’, %s, %s, %s, %d, %s”,
parent_no, child_no, s_date, e_date, unit_nam,
lead_time, chang_no, sup_ident, childs, quant);
 mw_insert(“bom_child_1”,help);
 }

/*****************************************************************************/
/* The function pop_bom_head gets the data from the bill_of_mat table and */
/* puts it into the bom_parent table. The function pop_child is call for */
/* all children of the specified part. */
/*****************************************************************************/

void pop_bom_head(bom_ident)
char *bom_ident;
{
 char help[LONG_STRING];
 FILE *drive;
 char file_rest[FILE_LINE_LEN];
 char loc_file[MEDIUM_STRING];

 char parent_no[SMAL_STRING], s_date[SMAL_STRING], e_date[SMAL_STRING];
 char unit_nam[SMAL_STRING], lead_time[SMAL_STRING], child_id[SMAL_STRING];
 int childs;

 sprintf(loc_file,”%s0”,temp_sel);

 printf(“\nIdent: %s.”,bom_ident);
 level_number = 0;

 /* Get data from the bill_of_mat table. */

 sprintf(help,”b.ident = %s”, bom_ident);
 mw_query(“bom_head”,help,””);
 next_value(result_string, parent_no, result_string);
 next_value(result_string, s_date, result_string);
 next_value(result_string, e_date, result_string);
 next_value(result_string, unit_nam, result_string);
 next_value(result_string, lead_time, result_string);

 /* Get all children. */

 sprintf(help,”bill_of_mat_ident=%s”,bom_ident);
 mw_query(“bill_of_mat_item_1”,help,loc_file);
 if((drive=fopen(loc_file,”r”)) == NULL)
printf(“\nError oper file %s.”,loc_file);
 file_rest[0]=‘\0’;
 childs = 0;

 /* Call pop_child for each child (if any). */
 while(file_reader(child_id, drive, file_rest))
 {
pop_child(child_id, 1);
childs++;
 }
 fclose(drive);

 /* Set non existant values to NULL. */

 if(e_date[0]) sprintf(e_date,"'%s'",e_date); else sprintf(e_date,"NULL");
 if(!lead_time[0]) sprintf(lead_time,"NULL");

 printf("\n%s, %s, %s, %s, %s, %d, %d",parent_no, s_date, e_date, unit_nam,
lead_time, childs, level_number);

 /* Inser row in the table bom_parent. */

 sprintf(help,"'%s', '%s', %s, '%s', %s, %d, %d",
parent_no, s_date, e_date, unit_nam,
lead_time, childs, level_number);
 mw_insert("bom_parent",help);
 }

/*************************************************************************/
/* The function start_pop_bom deletes all data in the tables bom_parent */
/* and bom_child and get calls pop_bom_head for all parts with a bom name */
/* 'BOM' and an item_no like 'LUT%'. */
/*************************************************************************/

void start_pop_bom()
{
 char help[LONG_STRING];
 FILE *drive;
 char file_rest[FILE_LINE_LEN];
 char bom_ident[SMAL_STRING];

 mw_delete("bom_parent","");
 mw_delete("bom_child_1","");

 sprintf(help,"item_no LIKE 'LUT%c' and nam = 'BOM'",'%'); /*don't change */
 mw_query("bill_of_mat_1",help,temp_sel);
 if((drive=fopen(temp_sel,"r")) == NULL)
 {
printf("\nError oper file %s.",temp_sel);
return;
 }
 file_rest[0] = '\0';
 while(file_reader(bom_ident, drive, file_rest))
pop_bom_head(bom_ident);
 fclose(drive);
}

main()
{
 init_oracle();
 start_pop_bom();
 terminate_oracle();
}
```

Meta-file definition

Name of meta-file : config_tables_bom.txt

Definition of database-objects: (for access_ora)

```
“bill_of_mat_1” “bill_of_mat”
“ident” ““‘

“bom_head” “bill_of_mat b, item_sourc_dat i”
“b.item_no, b.eff_from_dat, b.eff_to_dat, b.unit_nam, i.lead_tim”
“b.item_no = i.item_no (+)”

“bill_of_mat_item_1” “bill_of_mat_item”
“ident” ““‘

“bom_child” “bill_of_mat_item b, item_sourc_dat i”
“b.bill_of_mat_item_no, b.item_no, b.eff_from_dat, b.eff_to_dat, b.unit_nam, i.lead_tim,
b.eng_chang_ident, b.sup_ident, b.quant”
“b.item_no = i.item_no (+)”

“bom_parent” “bom_parent”
“parent_part_no, eff_start_date, eff_end_date, unit_of_measure, lead_time_offset,
number_of_children, number_of_levels” ““‘

“bom_child_1” “bom_child”
“parent_part_no, part_number, eff_start_date, eff_end_date, unit_of_measure,
lead_time_offset, eng_change_no, phases_out_part_no, number_of_children, quantity_per_assembly” ““‘
```

APPENDIX VII

MCS FIMM

Functional Interaction Manager - Service Options

	Functional Interaction Manager	
	Service Options	**Details**
Manipulation Services	**Add New Part**	Registers a new part to be processed. The number of the new part and its type must be given as an argument. The part number must be unique and is restricted to integers between 0 and 99999999. The type of a part can be any character string with a length of 3 alphanumeric characters. The status of all activities defined in the FIMM configuration data is set to pending ('P') for the new part and its instance is set to 1.
	Change Status	Change Status: Changes the status of a function for a given part, identified by its part number. If the new status of the function is complete ('C') the instance of the part is changed as well. The instance of the part will be set to the end instance of the function, as defined in the FIMM configuration data. When two or more functions have defined the same end instance, the instance of the part is changed only when all functions with the same end instance are completed. Otherwise a function, waiting for the competition of two other functions could get a part number on its job load list, while one of thefunctions is still working on that part (see also get job load).
	Change Instance	Changes the instance of a part, identified by its part number. Using this service a function can indicate to the Functional Interaction Manager that it has reached a certain status for a part, which might trigger other functions to start working on that part.
Query Services	**Get Parts**	Get Parts: Get information about parts, specified by four parameters. The first two parameters refer to the range of part numbers, the retrieved parts must lie in. If no lowest or highest number is specified, the values 0 and 99999999 are assumed as range. The other two parameters specify a function and a status value. The returned part numbers will only belong to parts which have the specified status for the given function. If no function is specified, all parts with numbers lying in the specified range are returned. A fifth parameter determines the format of the returned data. Either only the part numbers, or the complete status, instance and type information can be returned.
	Get Job Load	Get Job Load: Gets information about all parts, that have to be processed by a specified function. Besides the function, a range of part numbers can be specified, the retrieved parts should lie in. There are two conditions, that specify the returned parts. First, the status of the parts must be pending ('P') for the specified function. Second, the start instance of the function must be greater or equal than the actual instance of the part. The last parameter again determines the format of the returned data.

Program listings
for
Functional Interaction Manager & Engineering Resource Manager

```
/* File: operator_functions.c*/
/* The functions in this file perform tasks of the Functional Interaction Manager and
Engineering Resource Manager).*/

/* Functions include the following :*/
/*new_part()
 enters a new part to be registered in the MCS FIMM. The status for all activities is set to ‘P’ for pending
and the point of the part is set to 1.*/

/*change_status()
changes the status of an activity for a given part. If the new status of the activity is ‘C’ for completed,
 the point of the part is set to the end point of the activity. The point is not changed, if there are other
activities with the same end point and a status other than complete.*/

/*change_point()
sets the point of a given part to a new value.*/

/*add_eng_resource()
adds the location of an engineering resource to the eng_resource_table. If there was already a location given,
the old value is overwritten.*/

/*get_jobs()
retrieves parts fulfilling a given condition.*/

/*get_job_load()
retrives parts which are pending (‘P’) for a given activity and with point values greater or equal to the
start point of the given activity.*/

/*get_eng_resources()
retrieves the location of engineering resources.*/

/*get_resp_eng()
retrieves all engineering resources related to a given activity.*/


#include <stdio.h>
#include “local_incl.h”

extern char result_string[VERY_LONG_STRING];
char help[MEDIUM_STRING], help2[MEDIUM_STRING];
char val1[SMAL_STRING], val2[SMAL_STRING], val3[SMAL_STRING];
char value[SMAL_STRING], result_rest[MEDIUM_STRING];
FILE *drive, *drive2, *drive3;
char file_rest[FILE_LINE_LEN], file_rest2[FILE_LINE_LEN];
extern char temp_sel[MEDIUM_STRING];
extern char temp_sel2[MEDIUM_STRING];
extern char job_file[MEDIUM_STRING];
extern char resource_file[MEDIUM_STRING];

char new_part(part_no, type)
char *part_no;
char *type;
{
 int part_number;
 char part_number_string[SMAL_STRING];

/* part_no must be a number */
 strcpy(part_number_string, part_no);
 part_number = to_number(part_no);
 sprintf(part_no,”%d”,part_number);
 if(strcmp(part_no,part_number_string)) return(0);

 if((strlen(part_no)>PART_NO_LEN) ||
(strlen(type)>STATUS_LEN) || (!part_no[0]) || (!type[0])) return(0);

/* part_no must not exist */
 sprintf(help,”part_no=’%s’ and type=’type’”,part_no);
 mw_query(“status_table”,help,””);
 if(next_value(result_string,value,result_rest))
return(0);
 else
 {
sprintf(help,”’%s’, ‘type’, ‘%s’”,part_no,type);
mw_insert(“status_table”,help);
sprintf(help,”’%s’, ‘point’, ‘1’”,part_no);
mw_insert(“status_table”,help);
 mw_query(“s_act”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
{
 sprintf(help,”’%s’, ‘%s’, ‘P’”, part_no, val1);
 mw_insert(“status_table”,help);
}
fclose(drive);
mw_commit();
return(1);
 }
 }

char change_status(part_no, activity, status)
char *part_no;
char *activity;
char *status;
{
 int part_number;
 char part_number_string[SMAL_STRING];

/* part_no must be a number */
 strcpy(part_number_string, part_no);
 part_number = to_number(part_no);
 sprintf(part_no,”%d”,part_number);
 if(strcmp(part_no,part_number_string)) return(0);

 if((strlen(part_no)>PART_NO_LEN) ||
(strlen(activity)>FIC_NAME_LEN) ||
(strlen(status)>STATUS_LEN) || (!status[0])) return(0);
```

```
/* part_no must exist in the status_table */
 sprintf(help,"part_no='%s' and type='type'",part_no);
 mw_query("status_table",help,"");
 if(!next_value(result_string,value,result_rest))
return(0);

/* activity must be known */
 sprintf(help2,"name='%s'",activity);
 mw_query("s_act",help2,"");
 if(!next_value(result_string,value,result_rest))
return(0);
 else
 {
/* decide whether update or insert is needed */
 sprintf(help,"part_no='%s' and type='%s'",part_no, activity);
 sprintf(help2,"status='%s'",status);
 mw_query("status_table",help,"");
 if(next_value(result_string,value,result_rest))
  mw_update("status_table",help2,help);
 else
 {
 sprintf(help,"'%s','%s','%s'",part_no,activity,status);
 mw_insert("status_table",help);
 }
 if(result_string[0] != 's')
{ mw_rollback(); return(0); }
 else
 {
/* if status = C set point to end point of activity */
 if(!strcmp(status,"C"))
 {
 sprintf(help,"activity='%s' and type='e_p'",activity);
 mw_query("activity_info_info",help,"");
 if(next_value(result_string,value,result_rest))
 {
/* check whether other activities have the same end point and are not */
/* in the status C */
 sprintf(help,"status!='C' and part_no='%s' and type in(select activity from activity_info_table where information='%s' and type='e_p')",
 part_no,value);
 mw_query("status_table",help,"");
 if(!next_value(result_string,val1,result_rest))
 {
  sprintf(help,"part_no='%s' and type='point'",part_no);
  sprintf(help2,"status='%s'",value);
 mw_update("status_table",help2,help);
 }
 }
 else
 {
 printf("\nCan't find end point for activity %s!",activity);
 mw_rollback(); return(0);
 }
 }
 }
 mw_commit();
 return(1);
 }
```

```
}

char change_point(part_no, point)
char *part_no;
char *point;
{
 int part_number;
 char part_number_string[SMAL_STRING];

/* part_no must be a number */
 strcpy(part_number_string, part_no);
 part_number = to_number(part_no);
 sprintf(part_no,”%d”,part_number);
 if(strcmp(part_no,part_number_string)) return(0);

 if((strlen(part_no)>PART_NO_LEN) ||
(strlen(point)>STATUS_LEN) || (!point[0])) return(0);

/* part_no must exist */
 sprintf(help,”part_no='%s' and type='point'”,part_no);
 sprintf(help2,”status='%s'”,point);
 mw_update(“status_table”,help2,help);
 if(strcmp(result_string,”s:1”))
return(0);
 else
 {
 mw_commit();
 return(1);
 }
 }

char add_eng_resource(part_no, resource, location)
char *part_no;
char *resource;
char *location;
{
 int part_number;
 char part_number_string[SMAL_STRING];

/* part_no must be a number */
 strcpy(part_number_string, part_no);
 part_number = to_number(part_no);
 sprintf(part_no,”%d”,part_number);
 if(strcmp(part_no,part_number_string)) return(0);

 if((strlen(part_no)>PART_NO_LEN) ||
(strlen(resource)>FIC_NAME_LEN) ||
(strlen(location)>FIC_NAME_LEN) || (!location[0])) return(0);

/* part_no must exist in the status_table */
 sprintf(help,”part_no='%s' and type='type'”,part_no);
 mw_query(“status_table”,help,””);
 if(!next_value(result_string,value,result_rest))
return(0);

/* resource must be known */
 sprintf(help2,”name='%s'”,resource);
 mw_query(“s_er”,help2,””);
 if(!next_value(result_string,value,result_rest))
return(0);

 sprintf(help,”part_no=’%s’ and eng_resource=’%s’”, part_no, resource);
 mw_query(“eng_resource_table”, help, “”);
 if(next_value(result_string,value,result_rest))
 {
sprintf(help2,”location=’%s’”,location);
mw_update(“eng_resource_table”,help2,help);
 }
 else
 {
sprintf(help,”’%s’,’%s’,’%s’”,part_no, resource, location);
mw_insert(“eng_resource_table”,help);
 }
 mw_commit();
 return(1);
 }

void get_jobs(part_no1,part_no2,activity,status,format)
char *part_no1;
char *part_no2;
char *activity;
char *status;
char *format;
{
 int part_number;

/* part_no must be a number */
 part_number = to_number(part_no1);
 sprintf(part_no1,”%d”,part_number);

 part_number = to_number(part_no2);
 sprintf(part_no2,”%d”,part_number);

 sprintf(help,”part_no>=’%s’ and part_no<=’%s’”,
part_no1[0] == ‘\0’ ? “0” : part_no1,
part_no2[0] == ‘\0’ ? “99999999” : part_no2);
 if(activity[0])
 {
sprintf(help2,”%s and type=’%s’ and status=’%s’”,
help, activity, status);
 }
 else strcpy(help2,help);
 mw_query(“status_table_numbers”,help2,temp_sel);
 sprintf(help,”mv %s %s”,temp_sel, job_file);
 if(format[0])
system(help);
 else
 {
drive3=fopen(job_file,”w”);
fprintf(drive3,”\”%s\” \”%s\” \”%s\”\n”,”part_no”,”type”, “point”);
mw_query(“s_act”,””,temp_sel2);
if((drive2 = fopen(temp_sel2,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel2);
file_rest2[0] = ‘\0’;
while(file_reader(val3, drive2, file_rest2))
 {
```

```
 fprintf(drive3," \"%s\"\n",val3);
}
fprintf(drive3," \"*\"");
fclose(drive2);
if((drive = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
file_rest[0] = '\0';
while(file_reader(val1, drive, file_rest))
{
 fprintf(drive3,"\n\"%s\" ",val1);
 sprintf(help,"part_no='%s' and type='type'",
val1);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 fprintf(drive3,"\"%s\"\n",value);
 sprintf(help,"part_no='%s' and type='point'",
val1);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 fprintf(drive3,"\"%s\" ",value);
 if((drive2 = fopen(temp_sel2,"r")) == NULL)
printf("\nError, open %s.",temp_sel2);
 file_rest2[0] = '\0';
 while(file_reader(val2, drive2, file_rest2))
 {
 sprintf(help,"part_no='%s' and type='%s'",
val1, val2);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 fprintf(drive3,"\"%s\"\n",value);
 }
 fprintf(drive3," \"*\"");
 fclose(drive2);
}
fclose(drive);
fclose(drive3);
 }
}

void get_job_load(part_no1,part_no2,activity,format)
char *part_no1;
char *part_no2;
char *activity;
char *format;
{
 int part_number;

/* part_no must be a number */
 part_number = to_number(part_no1);
 sprintf(part_no1,"%d",part_number);

 part_number = to_number(part_no2);
 sprintf(part_no2,"%d",part_number);

 sprintf(help,"part_no>='%s' and part_no<='%s'",
part_no1[0] == '\0' ? "0" : part_no1,
part_no2[0] == '\0' ? "99999999" : part_no2);
 if(activity[0])
 {
sprintf(help2,"%s and type='%s' and status='P'",
help, activity);
 }
 else strcpy(help2,help);
 mw_query("status_table_numbers",help2,temp_sel2);

 sprintf(help,"activity='%s' and type='s_p'",activity);
 mw_query("activity_info_info",help,"");
 if(!next_value(result_string,val2,result_rest))
printf("\nError, can't get start point from activity %s",activity);

 drive2=fopen(temp_sel,"w");
 if((drive = fopen(temp_sel2,"r")) == NULL)
printf("\nError, open %s.",temp_sel2);
 file_rest[0] = '\0';
 while(file_reader(val1, drive, file_rest))
 {
sprintf(help,"part_no='%s' and type='point' and status>='%s'",
val1,val2);
mw_query("status_table_status",help,"");
if(next_value(result_string,value,result_rest))
 fprintf(drive2," \"%s\"\n",val1);
 }
 fclose(drive);
 fclose(drive2);

 sprintf(help,"mv %s %s",temp_sel, job_file);
 if(format[0])
system(help);
 else
 {
drive3=fopen(job_file,"w");
fprintf(drive3,"\"%s\" \"%s\" \"%s\"\n","part_no","type", "point");
mw_query("s_act","",temp_sel2);
if((drive2 = fopen(temp_sel2,"r")) == NULL)
printf("\nError, open %s.",temp_sel2);
file_rest2[0] = '\0';
while(file_reader(val3, drive2, file_rest2))
{
 fprintf(drive3," \"%s\"\n",val3);
}
fprintf(drive3," \"*\"");
fclose(drive2);
if((drive = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
file_rest[0] = '\0';
while(file_reader(val1, drive, file_rest))
{
 fprintf(drive3,"\n\"%s\" ",val1);
 sprintf(help,"part_no='%s' and type='type'",
val1);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 fprintf(drive3,"\"%s\"\n",value);
 sprintf(help,"part_no='%s' and type='point'",
val1);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 fprintf(drive3,"\"%s\"\n",value);
 if((drive2 = fopen(temp_sel2,"r")) == NULL)
printf("\nError, open %s.",temp_sel2);
 file_rest2[0] = '\0';
 while(file_reader(val2, drive2, file_rest2))
 {
 sprintf(help,"part_no='%s' and type='%s'",
val1, val2);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 fprintf(drive3,"\"%s\"\n",value);
 }
 fprintf(drive3," \"*\"");
 fclose(drive2);
}
fclose(drive);
fclose(drive3);
 }
}

void get_eng_resources(part_no1,part_no2,resource,format)
char *part_no1;
char *part_no2;
char *resource;
char *format;
{
 int part_number;

/* part_no must be a number */
 part_number = to_number(part_no1);
 sprintf(part_no1,"%d",part_number);

 part_number = to_number(part_no2);
 sprintf(part_no2,"%d",part_number);

 sprintf(help,"part_no>='%s' and part_no<='%s'",
part_no1[0] == '\0' ? "0" : part_no1,
part_no2[0] == '\0' ? "99999999" : part_no2);
 if(resource[0])
 {
sprintf(help2,"%s and eng_resource='%s'",
help, resource);
drive3=fopen(temp_sel,"w");
fprintf(drive3,"\"%s\"\n",resource);
fclose(drive3);
 }
 else
 {
mw_query("eng_resource_table_res",help,temp_sel);
strcpy(help2,help);
 }
 mw_query("eng_resource_table_numbers",help2,temp_sel2);

 drive3=fopen(resource_file,"w");
 if(!format[0])
 {
fprintf(drive3,"\"%s\"\n","part_no");
```

APPENDIX VIII ✓

FIMM Configurator - Service options

FIMM Configurator	
Service Options	**Details**
Add MCS function	Registers a new MCS function to be managed. The user will be asked to enter the start and end instance and the relations to the existing information models and engineering resources for the newly registered function.
Add information model	Registers a new information model, which is necessary to support part manufacture. The user will have to enter the I/O relations of the new information model to the existing functions.
Add engineering resource	Registers a new engineering resource to the system. The user will be asked to enter a function which is accountable for the newly registered engineering resource.
Delete MCS function	Deregisters an existing MCS function and purges all its related information, which includes configuration data about the function as well as all dynamic data concerning the status of the function with regard to processing the parts for manufacture.
Delete information model	Deregisters an existing information model and purges all related information from the system.
Delete engineering resource	Deregisters an existing engineering resource and purges all related configuration data from the system. Dynamic data about that engineering resource (i.e. the location of the resource for a specific part to be manufactured) will, however, not be deleted.
Change start/end instance	Changes the start and end instance for a function. The start instance is used, when the MCS function is ready to receive the job for part processing. The end instance is used to indicate the completion of the job by the function concerned.
Change Input/Output (I/O) association	Changes the I/O association relationship between MCS functions and information models. The user can choose to alter the following : * Single I/O relation between function and information model. * All I/O relations of a function to the information models. * All I/O relations of an information model to the functions.
Change accountability	Changes the accountability of MCS function for an engineering resource. Note that a function can be accountable for more than one engineering resource.
Display MCS functions	List all existing MCS functions.
Display information models	List all existing information models.
Display engineering resources	List all existing engineering resources.
Display I/O association	The following choices are available : * All I/O relations of a function to the information models. * All I/O relations of an information model to the functions.
Display status	Display status of functional interaction manager with regards to parts being processed.

Program listings FIMM Configurator

```
/* File: engineer.c */
/* Functions for FIMM Configurator */
/* This program has to be started on the host wayne (location of FIMM database). */
/
#include <stdio.h>
#include “local_incl.h”

extern char result_string[VERY_LONG_STRING];

main()
{
int i,j,k;
char command[SMAL_STRING];
char val1[SMAL_STRING], val2[SMAL_STRING], val3[SMAL_STRING];
char val4[SMAL_STRING], help[MEDIUM_STRING], help2[MEDIUM_STRING];
char value[SMAL_STRING], result_rest[MEDIUM_STRING];
FILE *drive, *drive2;
char file_rest[FILE_LINE_LEN], file_rest2[FILE_LINE_LEN];

char file_path[MEDIUM_STRING];
char temp_sel[MEDIUM_STRING];
char temp_sel2[MEDIUM_STRING];

strcpy(file_path,”/home2/sandra/valdew/misc”);
sprintf(temp_sel,”%s/temp_sel”,file_path);
sprintf(temp_sel2,”%s/temp_sel2”,file_path);

mw_connect(“mcc21”,”m”,”/home2/sandra/valdew/oracle_c/fim/config_tables.txt”);

printf(“\nEnter command (press RETURN for command list): “);
gets(command);
while((strcmp(command, “quit”))&&(command[0]!=’q’))
{
 if((command[0] == ‘n’)&&(command[1]==’a’))
  {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,””);
if(next_value(result_string,value,result_rest))
printf(“\nActivity %s exists allready.”, val1);
else
{
 sprintf(help,”’act’,‘%s’”,val1);
 mw_insert(“i_flat”,help);
 mw_query(“s_dm”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 printf(“\nEnter role of %s for %s (i, o, i/o, -): “,
val2, val1);
 gets(val3);
 sprintf(help,"'%s', ‘%s’, ‘%s’",val1, val2, val3);
 mw_insert(“in_out_table”,help);
 }
 fclose(drive);
 printf(“\nEnter start point: “);
 gets(val2);
 sprintf(help,"'%s', ‘s_p’, ‘%s’",val1,val2);
 mw_insert(“activity_info_table”,help);
 printf(“\nEnter end point: “);
 gets(val2);
 sprintf(help,"'%s', ‘e_p’, ‘%s’",val1,val2);
 mw_insert(“activity_info_table”,help);
 mw_query(“s_er”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 printf(“\nIs %s responsible for %s (y,n): “,
val1, val2);
 gets(val3);
 if(val3[0] == ‘y’)
 {
 sprintf(help,"'%s', ‘e_r’, ‘%s’",val1,val2);
 mw_insert(“activity_info_table”,help);
 }
 }
}
 }
 else if((command[0] == ‘d’)&&(command[1]==’a’))
  {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nActivity %s not known.”,val1);
else
{
 sprintf(help,”type=’act’ and name=’%s’”,val1);
 mw_delete(“i_flat”,help);
 sprintf(help,”type=’%s’”,val1);
 mw_delete(“status_table”,help);
 sprintf(help,”activity=’%s’”,val1);
 mw_delete(“in_out_table”,help);
 mw_delete(“activity_info_table”,help);
}
 }
 else if((command[0] == ‘p’)&&(command[1]==’a’))
  {
printf(“\n\n\nThe following activities are defined:\n”);
printf(“\n\t {Activity}”);
printf(“\n\t____________”);
mw_query(“s_act”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
```

```
printf(“\n\t%s”,val1);
printf(“\n\n”);
fclose(drive);
 }

else if((command[0] == ‘a’)&&(command[1]==’s’))
  {
printf(“\n\n\nThe following new activities are defined:\n”);
printf(“\n\t{Activity - IDEF0}”);
printf(“\n\t________________”);
mw_query(“s_act_status”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
printf(“\n\t%s”,val1);
printf(“\n\n”);
fclose(drive);
 }

 else if((command[0] == ‘p’)&&(command[1]==’r’))
  {
printf(“\nThe following engineering resources are defined:”);
mw_query(“s_er”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
printf(“\n\t%s”,val1);
fclose(drive);
 }
 else if((command[0] == ‘p’)&&(command[1]==’d’))
  {
printf(“\n\n\nThe following data models are defined:\n”);
printf(“\n\t{Data Model}”);
printf(“\n\t____________”);
mw_query(“s_dm”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
printf(“\n\t%s”,val1);
printf(“\n\n”);
fclose(drive);
 }

else if((command[0] == ‘d’)&&(command[1]==’s’))
  {
printf(“\n\n\nThe following new data models are defined:\n”);
printf(“\n\t{Data Model - IDEF1X}”);
printf(“\n\t___________________”);
mw_query(“s_dm_status”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
printf(“\n\t%s”,val1);
printf(“\n\n”);
```

```
fclose(drive);
 }


 else if((command[0] == ‘n’)&&(command[1]==’d’))
  {
printf(“\nEnter data model name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_dm”,help,””);
if(next_value(result_string,value,result_rest))
printf(“\nData model %s exists allready.”, val1);
else
{
 sprintf(help,”’dm’, ‘%s’”,val1);
 mw_insert(“i_flat”,help);
 mw_query(“s_act”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 printf(“\nEnter role of %s for %s (i, o, i/o, -): “,
val1, val2);
 gets(val3);
 sprintf(help,”’%s’, ‘%s’, ‘%s’”,val2, val1, val3);
 mw_insert(“in_out_table”,help);
 }
 fclose(drive);
}
 }
 else if((command[0] == ‘d’)&&(command[1]==’d’))
  {
printf(“\nEnter data model name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_dm”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nData model %s not known.”,val1);
else
{
 sprintf(help,”type=’dm’ and name=’%s’”,val1);
 mw_delete(“i_flat”,help);
 sprintf(help,”data_model=’%s’”,val1);
 mw_delete(“in_out_table”,help);
}
 }
 else if((command[0] == ‘n’)&&(command[1]==’r’))
  {
printf(“\nEnter engineering resource name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_er”,help,””);
if(next_value(result_string,value,result_rest))
printf(“\nEng. resource %s exists allready.”, val1);
else
{
 sprintf(help,”’er’, ‘%s’”,val1);
 mw_insert(“i_flat”,help);
 printf(“\nWhich activity is responsible for %s: “,val1);
 gets(val2);
 sprintf(help,”name=’%s’”,val2);
 mw_query(“s_act”,help,””);
 while((val2[0]!=’\0’) &&
(!next_value(result_string,value,result_rest)))
 {
 printf(“\nActivity %s is not known.”);
 printf(“ Try again or enter ‘RETURN’ to abort.”);
 printf(“\nWhich activity is responsible for %s: “,val1);
 gets(val2);
 sprintf(help,”name=’%s’”,val2);
 mw_query(“s_act”,help,””);
 }
 if(val2[0]!=’\0’)
 {
 sprintf(help,”’%s’,’e_r’,’%s’”,val2,val1);
 mw_insert(“activity_info_table”,help);
 }
}
 }
 else if((command[0] == ‘d’)&&(command[1]==’r’))
  {
printf(“\nEnter engineering resource name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_er”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nEng. resource %s not known.”,val1);
else
{
 sprintf(help,”type=’er’ and name=’%s’”,val1);
 mw_delete(“i_flat”,help);
 sprintf(help,”type=’e_r’ and information=’%s’”,val1);
 mw_delete(“activity_info_table”,help);
}
 }
 else if((command[0] == ‘s’)&&(command[1]==’a’))
  {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,”\0”);
if(!next_value(result_string,value,result_rest))
printf(“\nActivity %s not known.”,val1);
else
{
 printf(“\nInput data for activity %s:”,val1);
 mw_query(“s_dm”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 sprintf(help,”activity=’%s’ and data_model =’%s’”
,val1,val2);
 mw_query(“in_out_info”,help,””);
 if(next_value(result_string,value,result_rest))
 if(value[0] == ‘i’)
printf(“\n\t%s”,val2);
 }
 fclose(drive);
 printf(“\n\nOutput data for activity %s:”,val1);
 mw_query(“s_dm”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 sprintf(help,”activity=’%s’ and data_model =’%s’”
,val1,val2);
 mw_query(“in_out_info”,help,””);
 if(next_value(result_string,value,result_rest))
 if((value[0] == ‘o’)||(value[1] == ‘/’))
printf(“\n\t%s”,val2);
 }
 fclose(drive);
}
 }
 else if((command[0] == ‘s’)&&(command[1]==’d’))
  {
printf(“\nEnter data model name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_dm”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nData model %s not known.”,val1);
else
{
 printf(“\nActivities getting input from %s:”,val1);
 mw_query(“s_act”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 sprintf(help,”activity=’%s’ and data_model =’%s’”
,val2,val1);
 mw_query(“in_out_info”,help,””);
 if(next_value(result_string,value,result_rest))
 if(value[0] == ‘i’)
printf(“\n\t%s”,val2);
 }
 fclose(drive);
 printf(“\n\nActivities producing output for %s:”,val1);
 mw_query(“s_act”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 sprintf(help,”activity=’%s’ and data_model =’%s’”
,val2,val1);
 mw_query(“in_out_info”,help,””);
 if(next_value(result_string,value,result_rest))
 if((value[0] == ‘o’)||(value[1] == ‘/’))
printf(“\n\t%s”,val2);
 }
 fclose(drive);
}
 }
 else if((command[0] == ‘c’)&&(command[1]==’a’))
  {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nActivity %s not known.”, val1);
else
{
 sprintf(help,”status=’used’”);
 sprintf(help2,”name=’%s’”,val1);
 mw_update(“i_flat_status”,help,help2);
 mw_query(“s_dm”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 printf(“\nEnter role of %s for %s (i, o, i/o, -): “,
val2, val1);
 gets(val3);
 if(val3[0] != ‘\0’)
 {
 sprintf(help,”activity=’%s’ and data_model=’%s’”
,val1,val2);
 mw_query(“in_out_table”,help,””);
 if(next_value(result_string,value,result_rest))
 {
 sprintf(help2,”relation=’%s’”,val3);
 mw_update(“in_out_table”,help2,help);
 }
 else
 {
  sprintf(help,”’%s’, ‘%s’, ‘%s’”,val1, val2, val3);
 mw_insert(“in_out_table”,help);
 }
  }
 }
 fclose(drive);
}
 }
 else if((command[0] == ‘c’)&&(command[1]==’d’))
  {
printf(“\nEnter data model name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_dm”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nData model %s not known.”, val1);
else
{
 sprintf(help,”status=’used’”);
 sprintf(help2,”name=’%s’”,val1);
 mw_update(“i_flat_status”,help,help2);
 mw_query(“s_act”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 printf(“\nEnter role of %s for %s (i, o, i/o, -): “,
val1, val2);
 gets(val3);
 if(val3[0] != ‘\0’)
 {
 sprintf(help,”activity=’%s’ and data_model=’%s’”
,val2,val1);
 mw_query(“in_out_table”,help,””);
 if(next_value(result_string,value,result_rest))
{
 sprintf(help2,”relation=’%s’”,val3);
 mw_update(“in_out_table”,help2,help);
}
else
{
  sprintf(help,”’%s’, ‘%s’, ‘%s’”,val2, val1, val3);
 mw_insert(“in_out_table”,help);
}
 }
 }
 fclose(drive);
}
 }
 else if((command[0] == ‘s’)&&(command[1]==’s’))
 {
printf(“\n%65s”,”<----Start and end points---->”);
printf(“\n%15s: “,”Activities”);
for(i=1;i<20;i++) printf(“|%2d”,i);
mw_query(“s_act”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
{
 printf(“\n%15s: “,val1);
 sprintf(help,”activity=’%s’ and type=’s_p’”,val1);
 mw_query(“activity_info_info”,help,””);
 next_value(result_string,value,result_rest);
 i=to_number(value);
 for(j=1;j<i;j++) printf(“| “);
 sprintf(help,”activity=’%s’ and type=’e_p’”,val1);
 mw_query(“activity_info_info”,help,””);
 next_value(result_string,value,result_rest);
 k=to_number(value);
 i = k - i;
 for(j=0;j<i;j++) printf(“|**”);
 for(j=k;j<20;j++) printf(“| “);
}
fclose(drive);
```

```
printf(“\n\n”);
 }
 else if((command[0] == ‘s’)&&(command[1]==’r’))
 {
printf(“\nResponsibility for engineering resources:\n”);
printf(“\n%16s %s”,”Activities:”,”Engineering resources:\n”);
mw_query(“s_act”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
{
 printf(“\n%15s: “,val1);
 sprintf(help,”activity=’%s’ and type=’e_r’”,val1);
 mw_query(“activity_info_info”,help,temp_sel2);
 if((drive2 = fopen(temp_sel2,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel2);
 file_rest2[0] = ‘\0’;
 while(file_reader(val2, drive2, file_rest2))
 {
 printf(“%10s “,val2);
 }
 fclose(drive2);
}
fclose(drive);
printf(“\n\n”);
 }
 else if((command[0] == ‘s’)&&(command[1]==’i’))
 {
printf(“\n\t\t\t\t\tData Model assignment\n”);
printf(“\t\t\t\t\t____________________”);
mw_query(“s_dm”,””,temp_sel2);
if((drive2 = fopen(temp_sel2,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel2);
file_rest2[0] = ‘\0’;
i=0;
while(file_reader(val2, drive2, file_rest2))
{ ,
 ++i;
 printf(“\n\t\t\t\t\t%4d<--->%s”,i,val2);
}
fclose(drive2);
printf(“\n\n\n%50sIn/Output Table”);
printf(“\n%21s”,”Activity/Data Model:”);
mw_query(“s_dm”,””,temp_sel2);
if((drive2 = fopen(temp_sel2,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel2);
file_rest2[0] = ‘\0’;
i=0;
while(file_reader(val2, drive2, file_rest2))
{
 ++i;
 printf(“%4d”,i);
}
fclose(drive2);
mw_query(“s_act”,””,temp_sel);
if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
```

```
file_rest[0] = '\0';
while(file_reader(val1, drive, file_rest))
{
 printf("\n%20s:",val1);
 if((drive2 = fopen(temp_sel2,"r")) == NULL)
printf("\nError, open %s.",temp_sel2);
 file_rest2[0] = '\0';
 while(file_reader(val2, drive2, file_rest2))
 {
 sprintf(help,"activity='%s' and data_model='%s'",
val1, val2);
 mw_query("in_out_info",help,"");
 next_value(result_string,value,result_rest);
 printf("%4s",value);
 }
 fclose(drive2);
 }
printf("\n\n");
fclose(drive);
 }
 else if((command[0] == 's')&&(command[1]=='t'))
 {
printf("\nStarting at part_no: ");
gets(val1);
printf("\nEnding at part_no:");
gets(val2);
printf("\nStatus table:\n");
 printf("\n%16s","Part No:");
sprintf(help,"part_no>='%s' and part_no<='%s'",
 val1[0] == '\0' ? "0" : val1,
 val2[0] == '\0' ? "99999999" : val2);
mw_query("status_table_numbers",help,temp_sel);
if((drive = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
file_rest[0] = '\0';
while(file_reader(val3, drive, file_rest))
 {
 printf("%5s",val3);
 }
fclose(drive);
printf("\n\n%16s","Type:");
if((drive = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
file_rest[0] = '\0';
while(file_reader(val3, drive, file_rest))
 {
 sprintf(help,"part_no='%s' and type='type'",
val3);
 mw_query("status_table_status",help,"");
 next_value(result_string,value,result_rest);
 printf("%5s",value);
 }
fclose(drive);
 printf("\n%16s","Point:");
if((drive = fopen(temp_sel,"r")) == NULL)
printf("\nError, open %s.",temp_sel);
file_rest[0] = '\0';
while(file_reader(val3, drive, file_rest))
```

```
 {
 sprintf(help,"part_no='%s' and type='point'",
val3);
 mw_query(“status_table_status”,help,“”);
 next_value(result_string,value,result_rest);
 printf(“%5s”,value);
 }
fclose(drive);
printf(“\n”);
mw_query(“s_act”,“”,temp_sel2);
if((drive2 = fopen(temp_sel2,"r")) == NULL)
printf(“\nError, open %s.”,temp_sel2);
file_rest2[0] = ‘\0’;
while(file_reader(val3, drive2, file_rest2))
 {
 printf(“\n%15s:”,val3);
 if((drive = fopen(temp_sel,"r")) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
 while(file_reader(val2, drive, file_rest))
 {
 sprintf(help,"part_no='%s' and type='%s'",
val2, val3);
 mw_query(“status_table_status”,help,“”);
 next_value(result_string,value,result_rest);
 printf(“%5s”,value);
 }
 fclose(drive);
 }
fclose(drive2);
printf(“\n\n”);
 }
 else if((command[0] == ‘s’)&&(command[1]=='e'))
 {
printf(“\nStarting at part_no: “);
gets(val1);
printf(“\nEnding at part_no:”);
gets(val2);
printf(“\nengineering resource table:”);
printf(“\n%10s”,"part_no");
sprintf(help,"part_no>='%s' and part_no<='%s'",
val1[0] == ‘\0’ ? “0” : val1,
val2[0] == ‘\0’ ? “99999999” : val2);
mw_query(“eng_resource_table_res”,help,temp_sel2);
if((drive2 = fopen(temp_sel2,"r")) == NULL)
printf(“\nError, open %s.”,temp_sel2);.
file_rest2[0] = ‘\0’;
while(file_reader(val3, drive2, file_rest2))
{
 printf(“%10s”,val3);
}
fclose(drive2);
mw_query(“eng_resource_table_numbers”,help,temp_sel);
if((drive = fopen(temp_sel,"r")) == NULL)
printf(“\nError, open %s.”,temp_sel);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
{
 printf(“\n%9s:”,val1);
 if((drive2 = fopen(temp_sel2,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel2);
 file_rest2[0] = ‘\0’;
 while(file_reader(val2, drive2, file_rest2))
 {
 sprintf(help,”part_no=’%s’ and eng_resource=’%s’”,
val1, val2);
 mw_query(“eng_resource_table_location”,help,””);
 next_value(result_string,value,result_rest);
 printf(“%10s”,value);
 }
 fclose(drive2);
}
fclose(drive);
 }
 else if((command[0] == ‘c’)&&(command[1]==’p’))
 {
printf(“\nEnter part number: “);
gets(val1);
printf(“\nEnter activity name, ‘type’ or ‘point’: “);
gets(val2);
printf(“\nEnter new value: “);
gets(val3);
sprintf(help,”part_no=’%s’ and type=’%s’”,val1 ,val2);
sprintf(help2,”status=’%s’”,val3);
mw_query(“status_table”,help,””);
if(next_value(result_string, value,result_rest))
 mw_update(“status_table”,help2,help);
else
 printf(“\nEntry doesn’t exist. Nothing changed.”);
 }
 else if((command[0] == ‘c’)&&(command[1]==’r’))
 {
printf(“\nEnter part number: “);
gets(val1);
printf(“\nEnter engineering resource name: “);
gets(val2);
printf(“\nEnter new value: “);
gets(val3);
sprintf(help,”part_no=’%s’ and eng_resource=’%s’”,val1 ,val2);
sprintf(help2,”location=’%s’”,val3);
mw_query(“eng_resource_table”,help,””);
if(next_value(result_string, value,result_rest))
 mw_update(“eng_resource_table”,help2,help);
else
 printf(“\nEntry doesn’t exist. Nothing changed.”);
 }
 else if((command[0] == ‘c’)&&(command[1]==’s’))
 {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nActivity %s not known.”, val1);
else
{
 printf(“\nEnter new start point: “);
 gets(val2);
 sprintf(help,”activity=’%s’ and type=’s_p’”,val1);
 sprintf(help2,”information=’%s’”,val2);
 if(to_number(val2))
 mw_update(“activity_info_table”,help2,help);
 printf(“\nEnter new end point: “);
 gets(val2);
 sprintf(help,”activity=’%s’ and type=’e_p’”,val1);
 sprintf(help2,”information=’%s’”,val2);
 if(to_number(val2))
 mw_update(“activity_info_table”,help2,help);
}
 }
 else if((command[0] == ‘c’)&&(command[1]==’e’))
 {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nActivity %s not known.”, val1);
else
{
 printf(“\nEnter eng resource that must be provided by %s: “,
val1);
 gets(val2);
 sprintf(help,”name=’%s’”,val2);
 mw_query(“s_er”,help,””);
 if(!next_value(result_string,value,result_rest))
printf(“\nEng. resource %s not known.”, val2);
 else
 {
 sprintf(help,”’%s’, ‘e_r’, ‘%s’”,val1,val2);
 mw_insert(“activity_info_table”,help);
 }
}
 }
 else if((command[0] == ‘d’)&&(command[1]==’e’))
 {
printf(“\nEnter activity name: “);
gets(val1);
sprintf(help,”name=’%s’”,val1);
mw_query(“s_act”,help,””);
if(!next_value(result_string,value,result_rest))
printf(“\nActivity %s not known.”, val1);
else
{
 printf(“\nEnter eng resource that must not be provided by %s: “
,val1);
 gets(val2);
 sprintf(help,”name=’%s’”,val2);
 mw_query(“s_er”,help,””);
 if(!next_value(result_string,value,result_rest))
printf(“\nEng. resource %s not known.”, val2);
 else
 {
 sprintf(help,”activity=’%s’ and type=’e_r’ and information=’%s’”
,val1,val2);
 mw_delete("activity_info_table",help);
 }
}
 }
 else if((command[0] == 'c')&&(command[1]=='t'))
 {
mw_commit();
 }
 else if((command[0] == 'r')&&(command[1]=='b'))
 {
mw_rollback();
 }
 else
 {
 printf("\14\nAvailable commands are:");
 printf("\n 'pa' print activities");
 printf(" 'as' new activity - IDEF0");
 printf(" 'na' register new activity");
 printf(" 'da' delete activity");
 printf("\n 'pd' print data models");
 printf(" 'ds' new data model - IDEF1X");
 printf(" 'nd' register new data model");
 printf(" 'dd' delete data model");
 printf("\n\n 'pr' print engineering resource");
 printf("\n 'nr' new engineering resource");
 printf(" 'dr' delete engineering resource");
 printf("\n\n 'sa' show i/o for activity");
 printf(" 'sd' show i/o for data model");
 printf("\n 'si' show in/out table");
 printf("\n 'ca' change i/o for activity");
 printf(" 'cd' change i/o for data model");
 printf("\n\n 'ss' show start/end points");
 printf("\n 'cs' change start/end points for activity");
 printf("\n\n 'sr' show responsibity for resource");
 printf("\n 'ce' add responsibitity for activity");
 printf("\n 'de' delete responsibity for activity");
 printf("\n\n 'st' show status table");
 printf(" 'cp' change status table");
 printf("\n\n 'se' show engineering resource table");
 printf("\n 'cr' change engineering resource table");
 printf("\n\n 'ct' commit");
 printf("\n 'rb' rollback");
 printf("\n 'q' commit & quit");
 printf("\nCommand (%s) unknown. Try again.\n",command);
 }
 printf("\nEnter command (press RETURN for command list): ");
 gets(command);
}
printf("\n\n");
mw_commit();
mw_disconnect();
}
```

Meta-file definition

Name of meta-file : config_tables.txt

Definition of database-objects: (for access_ora)

“s_act” object name“flat_table” table name
“name”fields“type = ‘act’” where condition

“s_act_status” object name“flat_table” table name
“name”fields“(type, status) = ((‘act’, ‘new’))” where condition

“s_dm” object name“flat_table” table name
“name”fields“type = ‘dm’” where condition

“s_dm_status” object name“flat_table” table name
“name”fields“(type, status) = ((‘dm’, ‘new’))” where condition

“s_er” object name“flat_table” table name
“name”fields“type = ‘er’” where condition

“i_flat” object name“flat_table” table name
“type, name”fields‘“‘ where condition

“i_flat_status” object name“flat_table” table name
“status, name”fields‘“‘ where condition

“status_table” object name“status_table” table name
“part_no, type, status”fields‘“‘ where condition

“status_table_status” object name“status_table” table name
“status”fields‘“‘ where condition

“status_table_numbers” object name“status_table” table name
“distinct part_no”fields‘“‘ where condition

“eng_resource_table_location” object name“eng_resource_table” table name
“location”fields‘“‘ where condition

“eng_resource_table” object name“eng_resource_table” table name
“part_no, eng_resource, location”fields‘“‘ where condition

“eng_resource_table_numbers” object name“eng_resource_table” table name
“distinct part_no”fields‘“‘ where condition

“eng_resource_table_res” object name“eng_resource_table” table name
“distinct eng_resource”fields‘“‘ where condition

“in_out_table” object name“in_out_table” table name
“activity, data_model, relation”fields‘“‘ where condition

“in_out_info” object name“in_out_table” table name
“relation”fields‘“‘ where condition

“activity_info_table” object name“activity_info_table” table name
“activity, type, information”fields‘“‘ where condition

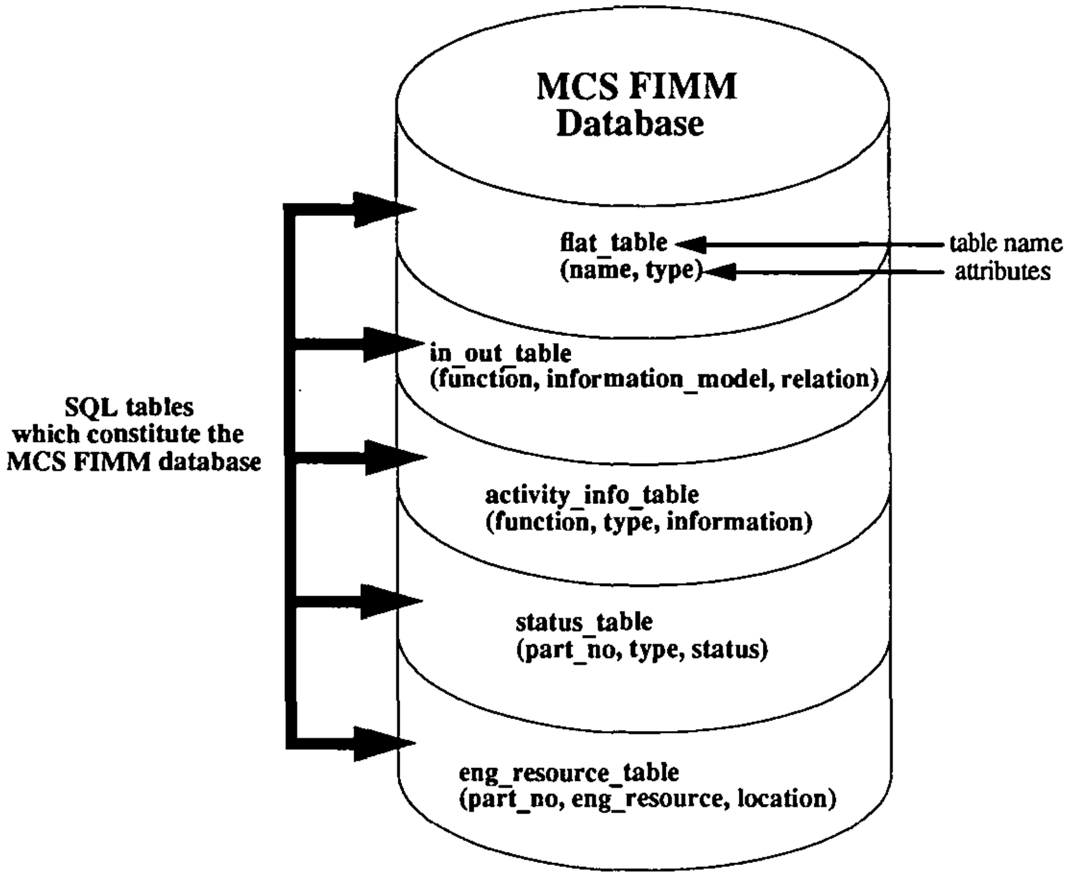
```
“activity_info_info” object name“activity_info_table” table name
“information”fields‘“‘ where condition

“bom” object name“bill_of_mat” table name
“ident”fields‘“‘ where condition

“rout” object name“rout” table name
“ident”fields‘“‘ where condition

“dem” object name“dem” table name
“ident”fields‘“‘ where condition

“ord” object name“ord_lin” table name
“int_ord_no”fields‘“‘ where condition

“bom_item” object name“bill_of_mat” table name
“item_no”fields‘“‘ where condition

“rout_item” object name“rout” table name
“item_no”fields‘“‘ where condition

“dem_item” object name“dem” table name
“item_no”fields‘“‘ where condition

“ord_item” object name“ord_lin” table name
“item_no”fields‘“‘ where condition

“mcc_reference”“‘mcc_reference”
“part_no, bill_of_mat_ident, rout_ident, dem_ident, order_no” ““‘

“get_bom”“‘mcc_reference, bill_of_mat bom”
“bom.item_no, bom.nam”“‘mcc_reference.bill_of_mat_ident = bom.ident”

“get_rout”“‘mcc_reference, rout r”
“r.item_no, r.nam”“‘mcc_reference.rout_ident = r.ident”

“get_dem”“‘mcc_reference m, dem d”
“d.ident, d.sch_ident”“‘m.dem_ident = d.ident”

“get_ord”“‘mcc_reference m, ord_lin o”
“o.item_no, o.int_ord_no”“‘m.order_no = o.int_ord_no”

“get_sch_job”“‘mcc_reference m, sch_job s”
“s.sch_ident, s.sch_job_no”“‘m.dem_ident = s.dem_ident”

“costed_bom” object name“costed_bom” table name
“item_no, quant, unit_nam, bill_of_mat_ident, typ, no_of_items” fields
““‘ where condition

“costed_bom_info”“‘costed_bom”
“quant, no_of_items” ““‘

“rout_bom_ident”“‘rout r, bill_of_mat b”
“r.bill_of_mat_ident, r.item_no, b.nam” “r.bill_of_mat_ident = b.ident”

“bill_of_mat_info”“‘bill_of_mat”
“ident, quant”“‘
```

“bill_of_mat_item_info”“bill_of_mat_item”
“item_no, quant, unit_nam, typ” ““‘

“dual” “dual”
“sysdate” ““‘

APPENDIX IX

MCS FIMM
Database Schema

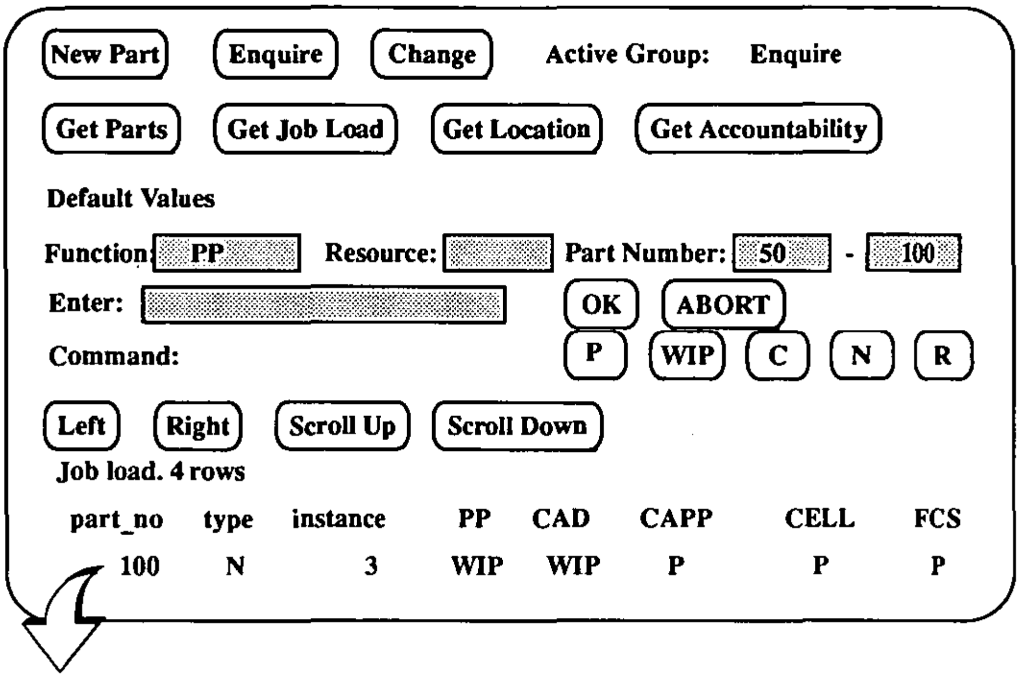


SQL Table	Stored data
flat_table	Functions to be managed, information entities and engineering resources.
in_out_table	Relations between functions and information entities.
activity_info_table	Start and end instance of functions and its accountability for engineering resources.
status_table	Dynamic information on status of functions for each and every part processed or being processed for manufacture.
eng_resource_table	Engineering resources and their location in relation to the parts being processed for manufacture.

APPENDIX X

Operational Characteristics and Services of MCS FIMM offered through Generic 'Application Shell'

User Interface Operation and Services of MCS FIMM



Active Group: Enquire

Default Values

Function: PP Resource: Part Number: 50 - 100

Enter:

Command:

Job load. 4 rows

part_no	type	instance	PP	CAD	CAPP	CELL	FCS
100	N	3	WIP	WIP	P	P	P

Display for part manufacture status

Procedure for service request

i. Chose a service group. This will present all available services in this group to the user.

ii. Chose the actual service.

iii. Enter all required parameters in the entry field as displayed in the command line. Press the 'OK' button to confirm your entry or 'ABORT' to choose another service.

iv. After entering the last parameter, the selected service is requested from the operation module.

v. The result of the service is displayed in a text field. Now the next service can be selected.

Default values are provided for function names, engineering resource names, part numbers ranges and single part numbers. For single part number, however, the default values are those of the first displayed part number retrieved from status or resource information normally displayed to the user.

Service groups available

- New Part
  This group consists of only one service which allows the user to register new parts to be processed for manufacture.
- Enquiry
  This group comprises all query services. They retrieve and display to the user status and engineering resource information from the FIMM database.
- Change
  These services allow the user to change status and instance data for specified parts and to enter engineering resource locations.

The following are details on the services available to the user :

New Part: This service requires two parameters and both cannot be chosen as default values. First the part number must be entered. This part number must be unique and not exist in the FIMM database. The second parameter is the type of the new part. The user can enter this value either by pressing one of the buttons 'N' for a 'new' or 'R' for a 'repeated' type, or enter any other type value in the entry field and press 'OK' afterwards.

Get Parts: There are four parameters that determine which parts are retrieved. The lower and upper boundary for the range of part numbers is taken from the default fields. If no default values are entered in those fields, the maximum range is chosen automatically. This is done, wherever a part number range is necessary for a service parameter and the user will never be asked to enter values for that range. The other two parameters refer to a function and a status value. When a function is entered in the default field the user will be asked to enter a status value. The retrieved parts will then have the specified status value for the default function. If no default function is given, all parts in the specified range are retrieved. The status value can either be entered by the entry field or by using one of the buttons 'P', 'WIP', 'C'.

Get Job Load: The range of part numbers for the job load is again specified by the default values, like in the service 'get parts'. The remaining parameter refers to an activity. If no default value is given, the user will be asked to enter a function name.

Get Location: The range of part number is specified as before. If a default value for an engineering resource is entered in the appropriate field, only locations for that engineering resource are retrieved by this service. If no default engineering resource is available, the user is asked to enter one, or he/she can press the 'OK' button without entering a value to retrieve the location of all engineering resources for the specified part number range.

Get Accountability: To retrieve all engineering resources that a function is accountable for, the user can either enter a default value for a function before requesting this service or enter a function name in the entry field.

Change Status: This service requires three parameters. Two of them, the function name and the part number are taken from the default fields if available. The last parameter is the new status value of the function for the actual part. This value can either be entered by the entry field or by using one of the buttons 'P', 'WIP', 'C'.

Change Instance: The part for which the instance should be changed is again taken from the default field. The new Instance must be entered by the user via the entry field.

Store Location: The engineering resource name and the part number are taken from the default fields as before. The user will be asked to enter the location for the specified engineering resource.

If one of the fields for the default values is empty and the value is required by a service, the user will be asked to enter the appropriate value in the entry field. When the user enters the required parameter values for the service requested, and subsequently decides not to execute this service, the 'ABORT' button can be use to cancel. While a requested service is executed by MCS FIMM, the line 'Working...' is displayed in the command field. During this time, no service can be requested.

Finally, the buttons 'Left', 'Right', 'Scroll Up' and 'Scroll Down' are used to move through the retrieved information.

APPENDIX XI

Program listings for Communication Mechanism for remote MCS FIMM services over LAN

```
/* File: new_mcc_remote_side.c*/
/* The Functions in this file communicate with a remote process on the host wayne
(where FIMM database resides)*/

/* Functions include :*/
/*init_remote_new_mcc()
starts the remote process on the host wayne(location of FIMM database).*/

/*quit_remote_new_mcc()
stops the remote process and removes the notifier object new_client from the notifier.*/

/*new_popen()
creates two pipes and a child process. The two pipes are used as stdin and stdout by the child process.*/

/*new_pclose()
closes the two pipes new_p1 and new_p2.*/

/*new_input_handler()
It is called by the notifier, whenever the remote process sends an output to the stdout.
Depending on the function_state, a return function is called, with the received string as an argument.
All other functions check the length of their arguments and send a command string to the remote process.
The variable function_state is set to the apropriate value.*/


#include <stdio.h>
#include <sunwindow/notify.h>
#include “local_incl.h”

static intnew_popen_pid;
static int new_p1[2];
static int new_p2[2];

static char new_client_object;
static Notify_client new_client;

void new_popen();
void new_pclose();

Notify_value new_input_handler();

char function_state[3];
char send_string[LONG_STRING];
int send_string_len;

void init_remote_new_mcc(cimbios)
char *cimbios;
{
 int i;
 char *file_path_exe;
 char help[MEDIUM_STRING];

 char *getenv();

 file_path_exe = getenv(“NEW_MCC_EXE”);

 function_state[0] = ‘\0’;

 new_client = (char *)&new_client_object;
 sprintf(help,"%s/new_mcc_ora_side",file_path_exe);
 new_popen(help);

 notify_set_input_func(new_client, new_input_handler, new_p2[0]);

 strcpy(function_state,"in");

 if(!cimbios[0])
 {
notify_start();
 new_pclose();
 }
}

void quit_remote_new_mcc(cimbios)
char *cimbios;
{
 write(new_p1[1],"\n",1);

 if(cimbios[0])
 {
new_pclose();
notify_remove(new_client);
 }
 else
 {
new_pclose();
notify_stop();
 }
}

void new_popen(cmd)
char*cmd;
{
if (pipe(new_p1) < 0)
printf("\nnew_p1 not created!!");
if (pipe(new_p2) < 0)
printf("\nnew_p2 not created!!");
if ((new_popen_pid = fork()) == 0)
{
 close(new_p1[1]);
 close(0);
 dup(new_p1[0]);
 close(new_p1[0]);

 close(new_p2[0]);
 close(1);
 dup(new_p2[1]);
 close(new_p2[1]);

 execl("/usr/ucb/rsh", "wayne", cmd, 0);

 _exit(1); /*disaster*/
}
if (new_popen_pid == -1)
 printf("\nchild process not created!!!");
close(new_p1[0]);
close(new_p2[1]);
}

void new_pclose()
{
close(new_p1[1]);
close(new_p2[0]);
}

Notify_value new_input_handler()
{
 send_string[0] = '\0';
 send_string_len = 0;
 read(new_p2[0],&send_string[send_string_len++],1);
 while(send_string[send_string_len-1] != '\n')
read(new_p2[0],&send_string[send_string_len++],1);
 send_string[send_string_len-1] = '\0';

 if(!strcmp(function_state,"gj"))
 {
function_state[0] = '\0';
return_get_jobs(send_string);
 }
 else if(!strcmp(function_state,"np"))
 {
function_state[0] = '\0';
return_new_part(send_string);
 }
 else if(!strcmp(function_state,"cs"))
 {
function_state[0] = '\0';
return_change_status(send_string);
 }
 else if(!strcmp(function_state,"cp"))
 {
function_state[0] = '\0';
return_change_point(send_string);
 }
 else if(!strcmp(function_state,"ar"))
 {
function_state[0] = '\0';
return_add_eng_resource(send_string);
 }
 else if(!strcmp(function_state,"in"))
 {
function_state[0] = '\0';
 result_init_new_mcc();
 }
 else if(!strcmp(function_state,"gr"))
 {
function_state[0] = '\0';
return_get_eng_resources(send_string);
 }
 else if(!strcmp(function_state,"gl"))
 {
function_state[0] = '\0';
return_get_job_load(send_string);
 }
 else if(!strcmp(function_state,"ge"))
 {
function_state[0] = '\0';
return_get_resp_eng(send_string);
 }
 else if(!strcmp(function_state,"sb"))
 {
function_state[0] = '\0';
return_set_bom(send_string);
 }
 else if(!strcmp(function_state,"gb"))
 {
function_state[0] = '\0';
return_get_bom(send_string);
 }
 else if(!strcmp(function_state,"st"))
 {
function_state[0] = '\0';
return_set_rout(send_string);
 }
 else if(!strcmp(function_state,"gt"))
 {
function_state[0] = '\0';
return_get_rout(send_string);
 }
 else if(!strcmp(function_state,"sd"))
 {
function_state[0] = '\0';
return_set_dem(send_string);
 }
 else if(!strcmp(function_state,"gd"))
 {
function_state[0] = '\0';
return_get_dem(send_string);
 }
 else if(!strcmp(function_state,"so"))
 {
function_state[0] = '\0';
return_set_ord(send_string);
 }
 else if(!strcmp(function_state,"go"))
 {
function_state[0] = '\0';
return_get_ord(send_string);
 }
 else if(!strcmp(function_state,"gs"))
 {
function_state[0] = '\0';
return_get_sch_job(send_string);
 }
 else if(!strcmp(function_state,"rs"))
 {
function_state[0] = '\0';
return_create_files();
 }
 else if(!strcmp(function_state,"rw"))
 {
function_state[0] = '\0';
return_read_wip();
 }
 else
 {
printf("\nError(new_mcc_remote_side).");
printf("\nGot message(%s) in function_state: (%s)!!",send_string,
function_state);
 }

 return(NOTIFY_DONE);
}

char get_jobs(part_no1, part_no2, activity, status, format)
char *part_no1;
char *part_no2;
char *activity;
char *status;
char *format;
{
 if(function_state[0] || (strlen(part_no1)>PART_NO_LEN) ||
(strlen(part_no2)>PART_NO_LEN) ||
(strlen(activity)>FIC_NAME_LEN) ||
(strlen(status)>STATUS_LEN)) return(0);

 strcpy(function_state,"gj");

 sprintf(send_string,":2:%s:%d:%s:%d:%s:%d:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no1), part_no1, strlen(part_no2), part_no2,
strlen(activity), activity, strlen(status), status,
strlen(format),format);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char get_job_load(part_no1, part_no2, activity, format)
char *part_no1;
char *part_no2;
char *activity;
char *format;
{
 if(function_state[0] || (strlen(part_no1)>PART_NO_LEN) ||
(strlen(part_no2)>PART_NO_LEN) ||
(strlen(activity)>FIC_NAME_LEN)) return(0);

 strcpy(function_state,"gl");

 sprintf(send_string,":2:%s:%d:%s:%d:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no1), part_no1, strlen(part_no2), part_no2,
strlen(activity), activity, strlen(format),format);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char get_resp_eng(activity)
char *activity;
{
 if(function_state[0] || (strlen(activity)>FIC_NAME_LEN)) return(0);
 strcpy(function_state,"ge");

 sprintf(send_string,":2:%s:%d:%s\n",function_state,
strlen(activity), activity);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char get_eng_resources(part_no1, part_no2, resource, format)
char *part_no1;
char *part_no2;
char *resource;
char *format;
{
 if(function_state[0] || (strlen(part_no1)>PART_NO_LEN) ||
(strlen(part_no2)>PART_NO_LEN) ||
(strlen(resource)>FIC_NAME_LEN)) return(0);

 strcpy(function_state,"gr");

 sprintf(send_string,":2:%s:%d:%s:%d:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no1), part_no1, strlen(part_no2), part_no2,
strlen(resource), resource, strlen(format),format);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char new_part(part_no, type)
char *part_no;
char *type;
{
 if(function_state[0] || (strlen(part_no)>PART_NO_LEN) ||
(strlen(type)>STATUS_LEN) || (!part_no[0]) || (!type[0])) return(0);

 strcpy(function_state,"np");

 sprintf(send_string,":2:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no), part_no, strlen(type), type);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char change_status(part_no, activity, status)
char *part_no;
char *activity;
char *status;
{
 if(function_state[0] || (strlen(part_no)>PART_NO_LEN) ||
(strlen(activity)>FIC_NAME_LEN) ||
(strlen(status)>STATUS_LEN) || (!status[0])) return(0);

 strcpy(function_state,"cs");

 sprintf(send_string,":2:%s:%d:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no), part_no, strlen(activity), activity,
strlen(status), status);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
```

```
}

char change_point(part_no, point)
char *part_no;
char *point;
{
 if(function_state[0] || (strlen(part_no)>PART_NO_LEN) ||
(strlen(point)>STATUS_LEN) || (!point[0])) return(0);

 strcpy(function_state,"cp");

 sprintf(send_string,":2:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no), part_no, strlen(point), point);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char add_eng_resource(part_no, resource, location)
char *part_no;
char *resource;
char *location;
{
 if(function_state[0] || (strlen(part_no)>PART_NO_LEN) ||
(strlen(resource)>FIC_NAME_LEN) ||
(strlen(location)>FIC_NAME_LEN) || (!location[0])) return(0);

 strcpy(function_state,"ar");

 sprintf(send_string,":2:%s:%d:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no), part_no, strlen(resource), resource,
strlen(location), location);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }


char set_bom(part_no, item_no, bom_nam)
char *part_no;
char *item_no;
char *bom_nam;
{
 strcpy(function_state,"sb");

 sprintf(send_string,":2:%s:%d:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no), part_no, strlen(item_no), item_no,
strlen(bom_nam), bom_nam);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char get_bom(part_no)
char *part_no;
{
 strcpy(function_state,"gb");

 sprintf(send_string,":2:%s:%d:%s\n",function_state,
strlen(part_no), part_no);
```

```
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char set_rout(part_no, item_no, rout_nam)
char *part_no;
char *item_no;
char *rout_nam;
{
 strcpy(function_state,”st”);

 sprintf(send_string,”:2:%s:%d:%s:%d:%s:%d:%s\n”,function_state,
strlen(part_no), part_no, strlen(item_no), item_no,
strlen(rout_nam), rout_nam);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char get_rout(part_no)
char *part_no;
{
 strcpy(function_state,”gt”);

 sprintf(send_string,”:2:%s:%d:%s\n”,function_state,
strlen(part_no), part_no);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char set_dem(part_no, ident)
char *part_no;
char *ident;
{
 strcpy(function_state,”sd”);

 sprintf(send_string,”:2:%s:%d:%s:%d:%s\n”,function_state,
strlen(part_no), part_no, strlen(ident), ident);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char get_dem(part_no)
char *part_no;
{
 strcpy(function_state,”gd”);

 sprintf(send_string,”:2:%s:%d:%s\n”,function_state,
strlen(part_no), part_no);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
}

char set_ord(part_no, ident)
char *part_no;
char *ident;
{
 strcpy(function_state,"so");

 sprintf(send_string,":2:%s:%d:%s:%d:%s\n",function_state,
strlen(part_no), part_no, strlen(ident), ident);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char get_ord(part_no)
char *part_no;
{
 strcpy(function_state,"go");

 sprintf(send_string,":2:%s:%d:%s\n",function_state,
strlen(part_no), part_no);
 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char get_sch_job(part_no)
char *part_no;
{
 strcpy(function_state,"gs");

 sprintf(send_string,":2:%s:%d:%s\n",function_state,
strlen(part_no), part_no);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char create_files()
{
 strcpy(function_state,"rs");

 sprintf(send_string,":2:%s\n",function_state);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }

char read_wip()
{
 strcpy(function_state,"rw");

 sprintf(send_string,":2:%s\n",function_state);

 write(new_p1[1],send_string,strlen(send_string));
 return(1);
 }
```

APPENDIX XII

Overview of $IDEF_0$ and $IDEF_{1X}$

IDEF$_0$

IDEF$_0$ is the technique for modelling functions or activities of the enterprise. It is a descendant of the Structured Analysis and Design Technique (SADT) developed by Ross [Ross 1977].

As illustrated, the building block of this modelling approach is the activity box. The activity box defines an activity, or function in the enterprise that is being modelled. The activity may be a decision making, or information conversion activity, or it may be a material conversion activity, or both. Inputs to the activity are shown at the left of the box. Inputs are items (material, informational) that are transformed by the activity. Outputs of the activity are shown at the right of the box. Outputs are the results of the activity acting on the inputs. Controls are shown entering the activity box from the top. A control is a condition that governs the performance of the activity. For example, a control may be a set of rules governing the activity or a condition that must exist before the activity can be done. Mechanisms enter the activity box from below. A mechanism is the means by which an activity is realised. For example, a mechanism may be a machine, a worker or any enabling element. Refer to Figure A for illustration.

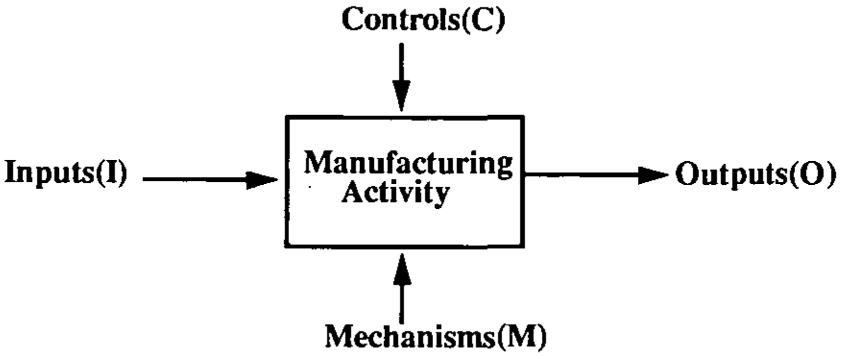


Figure A : The activity box ICOMs

IDEF$_0$ is applied using top down hierarchic decomposition. At the top of the hierarchy is the overall purpose of the model; it is the global activity that is the subject of the model. The overall activity is decomposable into components that, when taken together, comprise the global activity. This is the second tier of the architecture. Similarly, the second tier activities may be further decomposed into component activities. The decomposition process continues until there is sufficient detail to serve the purpose of the model builder. Refer to Figure B for illustration.

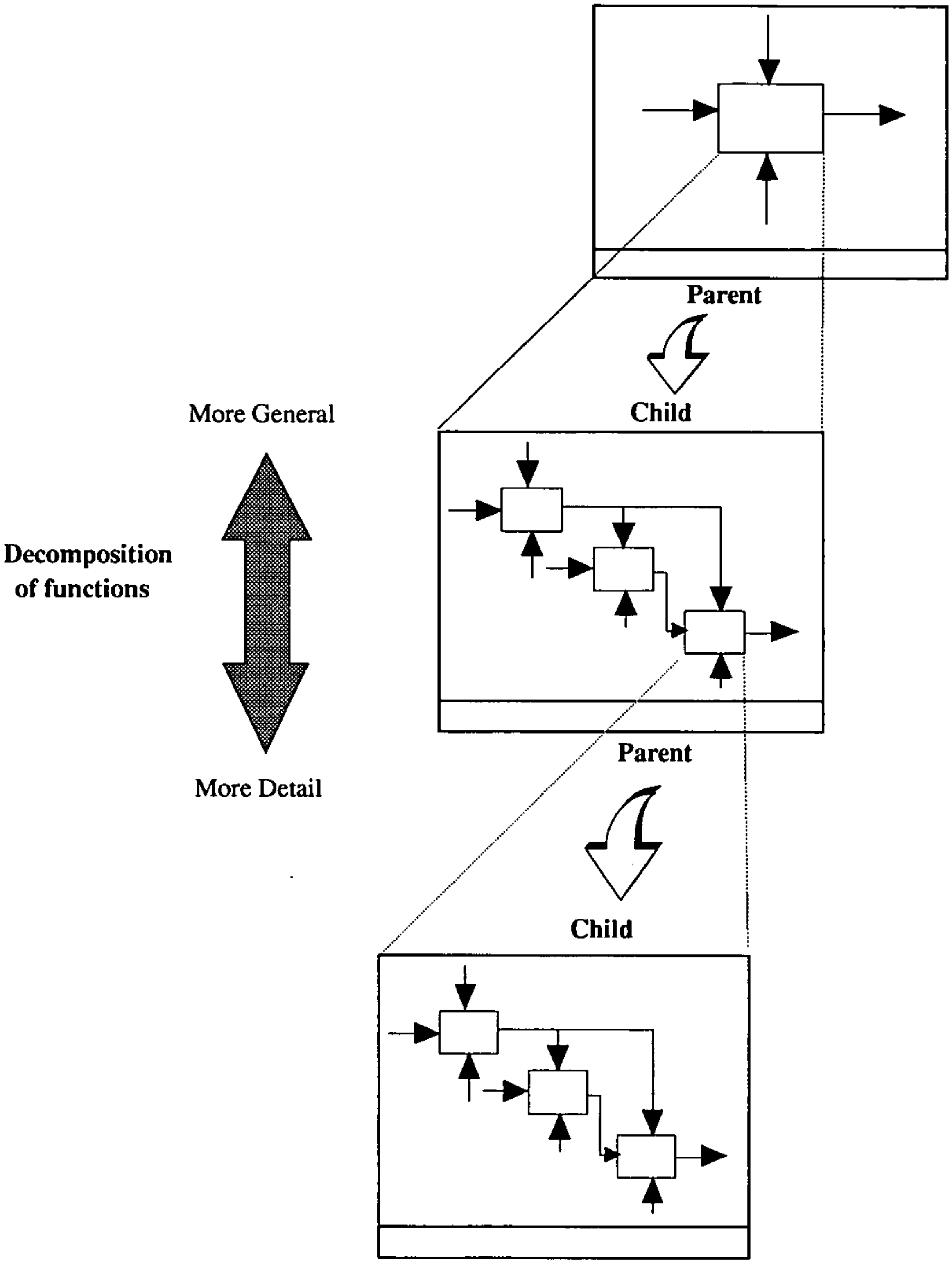


Figure B : Hierarchical decomposition of $IDEF_0$ model

IDEF$_{1X}$

IDEF$_{1X}$ is an extension of IDEF$_1$ and is for diagramming the information architechure. It is a semantic data modelling technique that defines the meaning of data within the context of its interrelationship with other data. IDEF$_{1X}$ uses the entity relationship approach based on the ER technique developed by Chen [Chen 1977]. A completed IDEF$_{1X}$ diagram is a static structure that defines information groupings and relationships among groupings.

As illustrated Figure C, the basic diagrammatic structure comprises boxes, which are used to represent entities. An entity is a set of real or abstract things (people, object, events) which have common attribute or characteristic. An attribute of an entity set, for which each instance must have a unique value is called a key attribute for that entity.

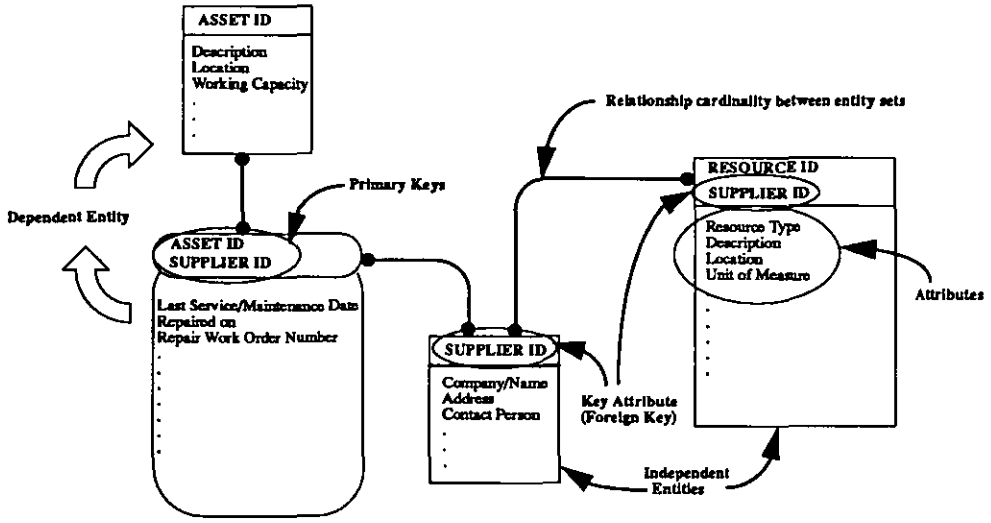


Figure C : IDEF$_{1X}$ Entity-Attribute relationship

In the IDEF1X diagram, entity attributes are listed with the box representing the entity. The key attribute is known as the primary key of a given entity and are separated from the rest of the attributes by a line that goes across the box. Relationships may exist between entities. A key attribute that provides the linkage between entities is called a foreign key. A relationship has cardinality which specifies the number of instances of an entity with which a given entity is associated through the relationship. There are the following possible relationships :

* One-to-One
* One-to-Many
* Many-to-Many

Each of the entities becomes a table in the database implementation. The set of attributes of each entity becomes an attribute field (or record field) of the entity table.

APPENDIX XIII

Report on
IDEF$_{1X}$ Entity-Attribute relationship model

{ENTITY, IE1, ORDER ENTRY, {MFG ORDER NUMBER,CUSTOMER ID,PARENT PART NUMBER}, {Preceeding Mfg Order Number,Description,Product Effectivity Start Date,Product Effectivity End Date,Type, Due Date,Unit of Measure,Unit Price,Order Quantity}}

{ENTITY, IE2, PART MASTER-BOM, {PARENT PART NUMBER,}; {Effectivity Start Date,Effectivity End Date,Unit of Measure,Engineering,Change Notice,Change Effected by,Date of Change,Phases out Part Number, Phased out by Part Number,Number of Levels,Number of components (children)}}

{ENTITY, IE4, PROCESS PLAN, {PROCESS PLAN ID,PART NUMBER,MFG CELL GROUP ID}, {Process Description,}}

{DEPENDENT ENTITY, DE21, BOM CHILD, {PARENT PART NUMBER,PART NUMBER}, {Number of components (children),Part Type,Quantity per Assembly,Effectivity Start Date,Effectivity End Date, Unit of Measure,Lead Time Offset,Engineering Change Notice,Change Effected by,Change Date,
Phases Out Part Number,Phased Out by Part Number}}

{DEPENDENT ENTITY, DE41 , MFG OPERATION ASSIGNMENT, {PROCESS PLAN ID,
MFG OPERATION ID}, {Mfg Operation Description,Preceeding Mfg Operation ID,Next Mfg Operation ID,
Alternative Mfg Operation ID,Setup time per item,Machining time per item,Handling time per item,
Operation time per item,Scrap rate,, }}

{DEPENDENT ENTITY, DE43, RESOURCE ASSIGNMENT, {MFG OPERATION ID,RESOURCE ID}, {Resource Type,Quantity Required,Unit of Measure}}

{DEPENDENT ENTITY, DE42, MFG FACILITY ASSIGNMENT, {MFG OPERATION ID,ASSET ID}, {Feed,Speed,Depth of cut,Number of passes,Remarks}}

{ENTITY, IE5, MFG CELL CONFIGURATION, {MFG CELL GROUP ID,ASSET ID}, {Number of Mfg Stations}}

{ENTITY, IE6, RESOURCE, {RESOURCE ID,SUPPLIER ID}, {Resource Type, Description,Location, Account Number,Unit of Measure,Unit Price,Buy/Make/Supply Code, Catalogue Order Number, Purchasing Lead Time, Last Order Date,Quantity Ordered, Effectivity Start Date,Effectivity End Date,Stock on-hand,Allocated/Reserved Stock,Scrap Value,Unit of Measure for Scrap}}

{ENTITY, IE3, ENGINEERING RESOURCE, {PART NUMBER}, {Engineering Resource, Location}}

{ENTITY, IE7, CUSTOMER, {CUSTOMER ID}, {Company/Name,Address,Contact Person,Telephone,Fax}}

{ENTITY, IE8, SCHEDULE, {MFG ORDER NUMBER,PART NUMBER}, {Priority,Order Status, Planned Quantity,Unit of Measure,Schedule Start Date,Schedule End Date}}

{ENTITY, IE9, SHOP FLOOR STATUS, {MFG ORDER NUMBER,PART NUMBER}, {Actual Quantity Produced,Work Centre/Cell Utilisation Rate,Actual Capacity Utilised}}

{ENTITY, IE10, MANUFACTURING FACILITY, {ASSET ID,}, {Description, Location, Working Capacity,Labor Cost per hour,Handling Cost per hour,,}}

{ENTITY, IE11, SUPPLIER, {SUPPLIER ID}, {Company/Name,Address,Contact Person,Telephone,Fax}}

{DEPENDENT ENTITY, DE101, PERSONNEL, { ASSET ID}, {Personnel ID,Name, Address, Telephone, Salary,Skill,Skill level,Remarks}}

{DEPENDENT ENTITY, DE102, MACHINE, {ASSET ID,SUPPLIER ID}, {Last Service/Maintenance Date, Repaired on,Repair Work Order Number,Max. job size accommodated - X/Y/Z axes,Accuracy,Machining Cost per hour,Horse Power,Speed Range (Max./Min.),Feed Range,Payload, Working Envelope - X/Y/Z/A/B axes,Setup Time,Tool Change Time,Feed Change Time,Table Rotation Time, Tool Adjustment Time,Rapid Tranverse Rate}}

{CATEGORIZATION, CR1,, COMPLETE, {IE10}, {DE101,DE102}

{RELATION, RL1,, NON-SPECIFIC, IE2, DE21, OM}

{RELATION, RL2,, NON-SPECIFIC, IE4, DE21, O}

{RELATION, RL3,, NON-SPECIFIC, DE41 , DE43, OM}

{RELATION, RL4,, NON-SPECIFIC, IE1, IE2, O}

{RELATION, RL5,, NON-SPECIFIC, IE4, DE41 , O}

{RELATION, RL6,, NON-SPECIFIC, DE41 , DE42, OM}

{RELATION, RL7,, NON-SPECIFIC, DE43, IE6, O}

{RELATION, RL8,, NON-SPECIFIC, IE1, IE7, O}

{RELATION, RL9,, NON-SPECIFIC, IE8, IE9, O}

{RELATION, RL10,, NON-SPECIFIC, IE9, DE21, O}

{RELATION, RL11,, NON-SPECIFIC, IE8, DE21, O}

{RELATION, RL12,, NON-SPECIFIC, IE4, IE5, ZOM}

```
{RELATION, RL13, , NON-SPECIFIC, IE3, IE4, O}

{RELATION, RL14, , NON-SPECIFIC, IE1, IE8, OM}

{RELATION, RL15, , NON-SPECIFIC, IE1, IE9, OM}

{RELATION, RL16, , NON-SPECIFIC, IE5, IE10, OM}

{RELATION, RL17, , NON-SPECIFIC, IE11, IE6, OM}

{RELATION, RL18, , NON-SPECIFIC, DE102, IE11, O}

{RELATION, RL19, , NON-SPECIFIC, DE21, IE3, OM}

{RELATION, RL20, , NON-SPECIFIC, DE42, IE10, O}


Done.
```

APPENDIX XIV

1. **EXPRESS based information model schema**

2. **EXPRESS index datafile**
   (assignment of unique identifier to entities in EXPRESS model)

3. **EXPRESS model data dictionary**

4. **EXPRESS to SQL Compiler generated datafile**
   (SQL commands to enable creation of relational database tables)

5. **Example of relational database tables relationship**
   (via the datafile generated by EXPRESS to SQL Complier)

1. EXPRESS based information model schema

```
SCHEMA INFORMATION_MODELS;

TYPE
type_class = enumeration of (repeat_order, one_off);
 END_TYPE;
 TYPE
part_class = enumeration of (normal, phantom, resource, co-product, tool, tool_return_item);
 END_TYPE;
 TYPE
resource_class = enumeration of (tool, tool_assessories, materials, fixtures,
fixture_assessories, miscellaneous);
 END_TYPE;
 TYPE
order_category = enumeration of (quote_forecast, open_order, confirmed_order,
closed_order_proceed,order_complete, closed_order_archive, order_purge, order_hold);
 END_TYPE;
 TYPE
product_code = enumeration of (make, buy, supply);
 END_TYPE;


 ENTITY ORDER_ENTRY;
 has_PART_MASTER-BOM : PART_MASTER-BOM;
 has_CUSTOMER : CUSTOMER;
 has_SCHEDULE : LIST [1:?] OF SCHEDULE;
 has_SHOP_FLOOR_STATUS : LIST [1:?] OF SHOP_FLOOR_STATUS;
Mfg_Order_Number  : INTEGER(7);
 Preceeding_Order_Number  : INTEGER(7);
 Description : STRING(60);
 Effectivity_Start_Date  : date;
 Effectivity_End_Date  : date;
 Type : type_class;
 Due_Date : date;
 Unit_of_Measure : STRING(4);
 Unit_Price : REAL;
 Order_Quantity : INTEGER(9);
 END_ENTITY;

 ENTITY PART_MASTER-BOM;
 has_BOM_CHILD : LIST [1:?] OF BOM_CHILD;
 Parent_part_number : STRING(15);
 Effectivity_Start_Date : date;
 Effectivity_End_Date : date;
 Unit_of_Measure : STRING(4);
 Engineering_Change_Notice : INTEGER(7);
 Change_Effected_by : STRING(20);
 Date_of_Change : date;
 Phases_out_Part_Number : STRING(15);
 Phased_out_by_Part_Number : STRING(15);
 Number_of_Levels : INTEGER(2);
 Number_of_components : INTEGER(3);
 END_ENTITY;

 ENTITY PROCESS_PLAN;
 has_BOM_CHILD : BOM_CHILD;
 has_MFG_OPERATION_ASSIGNMENT : MFG_OPERATION_ASSIGNMENT;
 has_MFG_CELL_CONFIGURATION : MFG_CELL_CONFIGURATION;
 Process_plan_ID : INTEGER(7);
Process_Description : STRING(60);
 END_ENTITY;
```

```
ENTITY BOM_CHILD;
has_ENGINEERING_RESOURCE : LIST [1:?] OF ENGINEERING_RESOURCE;
Part_number : STRING(15);
Number_of_components : INTEGER(3);
Part_Type : part_class;
Quantity_per_Assembly : REAL;
Effectivity_Start_Date : date;
Effectivity_End_Date : date;
Unit_of_Measure : STRING(4);
Lead_Time_Offset : INTEGER(9);
Engineering_Change_Notice : INTEGER(7);
Change_Effected_by : STRING(20);
Change_Date : date;
Phases_Out_Part_Number : STRING(15);
Phased_Out_by_Part_Number : STRING(15);
END_ENTITY;

ENTITY MFG_OPERATION_ASSIGNMENT;
has_RESOURCE_ASSIGNMENT : LIST [1:?] OF RESOURCE_ASSIGNMENT;
has_MFG_FACILITY_ASSIGNMENT : LIST [1:?] OF MFG_FACILITY_ASSIGNMENT;
Mfg_Operation_ID : INTEGER(7);
Mfg_Operation_Description : STRING(60);
Preceed_Mfg_Operation_ID  : INTEGER(7);
Next_Mfg_Operation_ID : INTEGER(7);
Alternate_Mfg_Operation_ID : INTEGER(7);
Setup_time_per_item : REAL;
Machining_time_per_item : REAL;
Handling_time_per_item : REAL;
Operation_time_per_item : REAL;
Scrap_rate : REAL;
END_ENTITY;

ENTITY RESOURCE_ASSIGNMENT;
has_RESOURCE : RESOURCE;
Resource_Type : resource_class;
Quantity_Required : REAL;
Unit_of_Measure : STRING(4);
END_ENTITY;

ENTITY MFG_FACILITY_ASSIGNMENT;
has_MANUFACTURING_FACILITY : MANUFACTURING_FACILITY;
Feed : INTEGER(4);
Speed : INTEGER(6);
Depth_of_cut : INTEGER(3);
Number_of_passes : INTEGER(5);
Remarks : STRING(60);
END_ENTITY;


ENTITY MFG_CELL_CONFIGURATION;
has_MANUFACTURING_FACILITY : LIST [1:?] OF MANUFACTURING_FACILITY;
Mfg_Cell_Group_ID : INTEGER(2);
Number_of_Mfg_Stations : INTEGER(2);
Mfg_station_1 : INTEGER(7);
Description_station_1 : STRING(60);
Mfg_station_2 : INTEGER(7);
Description_station_2 : STRING(60);
Mfg_station_3  : INTEGER(7);
Description_station_3 : STRING(60);
Mfg_station_4 : INTEGER(7);
Description_station_4 : STRING(60);
Mfg_station_5 : INTEGER(7);
Description_station_5 : STRING(60);
END_ENTITY;
```

```
ENTITY RESOURCE;
Resource_ID : STRING(15);
Resource_Type : resource_class;
Description : STRING(60);
Location : STRING(15);
Account_Number : INTEGER(15);
Unit_of_Measure : STRING(4);
Unit_Price : REAL;
Buy_Make_Supply_Code : product_code;
Catalogue_Order_Number : STRING(30);
Purchasing_Lead_Time : INTEGER(7);
Last_Order_Date : date;
Quantity_Ordered : INTEGER(9);
Effectivity_Start_Date : date;
Effectivity_End_Date : date;
Stock_on_hand : INTEGER(9);
Allocated_Reserved_Stock : INTEGER(9);
Scrap_Value : REAL;
Scrap_unit_of_Measure : STRING(4);
END_ENTITY;

ENTITY ENGINEERING_RESOURCE;
has_PROCESS_PLAN : PROCESS_PLAN;
Engineering_Resource : STRING(10);
Location : STRING(10);
END_ENTITY;

ENTITY CUSTOMER;
Customer_ID : INTEGER(7);
Company_Name : STRING(40);
Address : STRING(60);
Contact_Person : STRING(25);
Telephone : STRING(20);
Fax : STRING(20);
END_ENTITY;

ENTITY SCHEDULE;
has_SHOP_FLOOR_STATUS : SHOP_FLOOR_STATUS;
has_BOM_CHILD : BOM_CHILD;
Priority : INTEGER(3);
Order_Status : order_category;
Planned_Quantity : INTEGER(9);
Unit_of_Measure : STRING(4);
Schedule_Start_Date : date;
Schedule_End_Date : date;
END_ENTITY;

ENTITY SHOP_FLOOR_STATUS;
has_BOM_CHILD : BOM_CHILD;
Actual_Quantity_Produced : INTEGER(9);
Station_Utilisation_Rate  : REAL;
Actual_Capacity_Utilised : REAL;
END_ENTITY;

ENTITY MANUFACTURING_FACILITY
SUPERTYPE OF (ONEOF(PERSONNEL, MACHINE));
Asset_ID : INTEGER(7);
Description : STRING(60);
Location : STRING(15);
Working_Capacity : INTEGER(3);
Labor_Cost_per_hour : REAL;
Handling_Cost_per_hour : REAL;
END_ENTITY;
```

```
ENTITY SUPPLIER;
has_RESOURCE : LIST [1:?] OF RESOURCE;
Supplier_ID : INTEGER(7);
Company_Name : STRING(40);
Address : STRING(60);
Contact_Person : STRING(25);
Telephone : STRING(20);
Fax : STRING(20);
END_ENTITY;

ENTITY PERSONNEL
SUBTYPE OF (MANUFACTURING_FACILITY);
Personnel_ID : STRING(15);
Name : STRING(30);
Address : STRING(60);
Telephone : STRING(20);
Salary : REAL;
Skill : STRING(30);
Skill_level : INTEGER(2);
Remarks : STRING(60);
END_ENTITY;

ENTITY MACHINE
SUBTYPE OF (MANUFACTURING_FACILITY);
has_SUPPLIER : SUPPLIER;
Last_Service_Maintenance_Date : date;
Repaired_on : date;
Repair_Work_Order_Number : INTEGER(7);
Max_job_size_X_axis   : REAL;
Max_job_size_Y_axis  : REAL;
Max_job_size_Z_axis  : REAL;
Accuracy : REAL;
Machining_Cost_per_hour : REAL;
Horse_Power : INTEGER(7);
Speed_Range_Min : INTEGER(6);
Speed_Range_Max : INTEGER(6);
Feed_Range_Min : INTEGER(4);
Feed_Range_Max : INTEGER(4);
Payload : INTEGER(5);
Working_Envelope_X_axis  : REAL;
Working_Envelope_Y_axis  : REAL;
Working_Envelope_Z_axis  : REAL;
Working_Envelope_A_axis  : REAL;
Working_Envelope_B_axis  : REAL;
Setup_Time : REAL;
Tool_Change_Time : REAL;
Feed_Change_Time : REAL;
Table_Rotation_Time : REAL;
Tool_Adjustment_Time : REAL;
Rapid_Tranverse_Rate : REAL;
END_ENTITY;

ENTITY date;
day  : INTEGER(2);
month : INTEGER(2);
year : INTEGER(2);
END_ENTITY;


END_SCHEMA;
```

2. EXPRESS index datafile

```
e1:ORDER_ENTRY:
e2:ORDER_ENTRY_has_PART_MASTER-BOM:
e3:ORDER_ENTRY_has_CUSTOMER:
e4:ORDER_ENTRY_has_SCHEDULE:
e5:ORDER_ENTRY_has_SHOP_FLOOR_STATUS:
e6:ORDER_ENTRY_Effectivity_Start_Date:
e7:ORDER_ENTRY_Effectivity_End_Date:
e8:ORDER_ENTRY_Type:
e9:ORDER_ENTRY_Type:
e10:ORDER_ENTRY_Due_Date:
en0:PART_MASTER-BOM:
e11:PART_MASTER-BOM_has_BOM_CHILD:
e12:PART_MASTER-BOM_Effectivity_Start_Date:
e13:PART_MASTER-BOM_Effectivity_End_Date:
e14:PART_MASTER-BOM_Date_of_Change:
e15:PROCESS_PLAN:
e16:PROCESS_PLAN_has_BOM_CHILD:
e17:PROCESS_PLAN_has_MFG_OPERATION_ASSIGNMENT:
e18:PROCESS_PLAN_has_MFG_CELL_CONFIGURATION:
en5:BOM_CHILD:
e19:BOM_CHILD_has_ENGINEERING_RESOURCE:
e20:BOM_CHILD_Part_Type:
e21:BOM_CHILD_Effectivity_Start_Date:
e22:BOM_CHILD_Effectivity_End_Date:
e23:BOM_CHILD_Change_Date:
en6:MFG_OPERATION_ASSIGNMENT:
e24:MFG_OPERATION_ASSIGNMENT_has_RESOURCE_ASSIGNMENT:
e25:MFG_OPERATION_ASSIGNMENT_has_MFG_FACILITY_ASSIGNMENT:
en9:RESOURCE_ASSIGNMENT:
e26:RESOURCE_ASSIGNMENT_has_RESOURCE:
e27:RESOURCE_ASSIGNMENT_Resource_Type:
e28:RESOURCE_ASSIGNMENT_Resource_Type:
en10:MFG_FACILITY_ASSIGNMENT:
e29:MFG_FACILITY_ASSIGNMENT_has_MANUFACTURING_FACILITY:
en7:MFG_CELL_CONFIGURATION:
e30:MFG_CELL_CONFIGURATION_has_MANUFACTURING_FACILITY:
en11:RESOURCE:
e31:RESOURCE_Resource_Type:
e32:RESOURCE_Buy_Make_Supply_Code:
e33:RESOURCE_Last_Order_Date:
e34:RESOURCE_Effectivity_Start_Date:
e35:RESOURCE_Effectivity_End_Date:
en8:ENGINEERING_RESOURCE:
e36:ENGINEERING_RESOURCE_has_PROCESS_PLAN:
en1:CUSTOMER:
en2:SCHEDULE:
```

e37:SCHEDULE_has_SHOP_FLOOR_STATUS:

e38:SCHEDULE_has_BOM_CHILD:

e39:SCHEDULE_Order_Status:

e40:SCHEDULE_Order_Status:

e41:SCHEDULE_Schedule_Start_Date:

e42:SCHEDULE_Schedule_End_Date:

en3:SHOP_FLOOR_STATUS:

e43:SHOP_FLOOR_STATUS_has_BOM_CHILD:

en12:MANUFACTURING_FACILITY:

e44:SUPPLIER:

e45:SUPPLIER_has_RESOURCE:

e46:PERSONNEL:

e47:MACHINE:

e48:MACHINE_has_SUPPLIER:

e49:MACHINE_Last_Service_Maintenance_Date:

e50:MACHINE_Repaired_on:

en4:date:

t1:type_class:

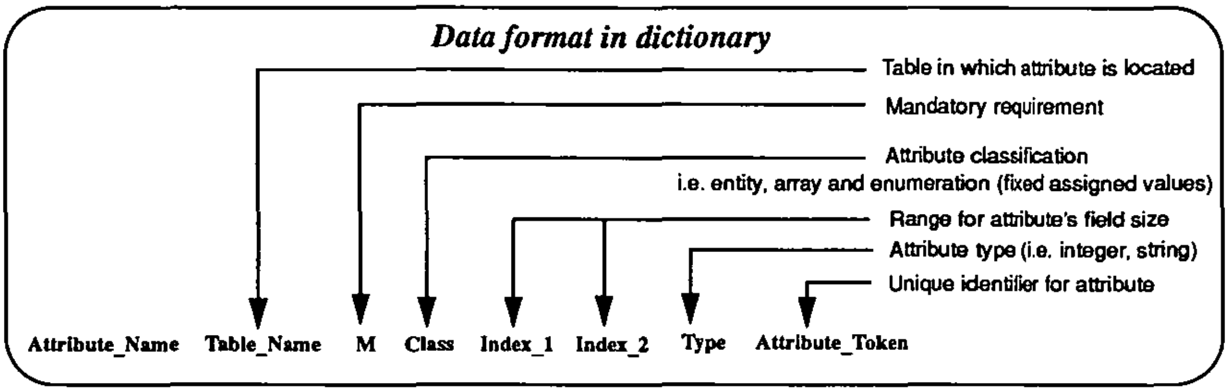
t2:part_class:

t3:resource_class:

t4:order_category:

t5:product_code:

3. EXPRESS model data dictionary



Data format in dictionary

Table in which attribute is located

Mandatory requirement

Attribute classification
i.e. entity, array and enumeration (fixed assigned values)

Range for attribute's field size

Attribute type (i.e. integer, string)

Unique identifier for attribute

Attribute_Name Table_Name M Class Index_1 Index_2 Type Attribute_Token

ORDER ENTRY

has_PART_MASTER-BOM e1 m entity en0 a0
has_CUSTOMER e1 m entity en1 a1
has_SCHEDULE e1 m LIST 1 # en2 a2
has_SHOP_FLOOR_STATUS e1 m LIST 1 # en3 a3

Mfg_Order_Number e1 m 1 7 INTEGER a4
Preceeding_Order_Number e1 m 1 7 INTEGER a5
Description e1 m 1 60 STRING a6
Effectivity_Start_Date e1 m entity en4 a7
Effectivity_End_Date e1 m entity en4 a8
Type e1 m enumeration t1 a9
Type e1 m entity t1 a10
Due_Date e1 m entity en4 a11
Unit_of_Measure e1 m 1 4 STRING a12
Unit_Price e1 m REAL a13
Order_Quantity e1 m 1 9 INTEGER a14

PART MASTER/BOM

has_BOM_CHILD en0 m LIST 1 # en5 a15

Parent_part_number en0 m 1 15 STRING a16
Effectivity_Start_Date en0 m entity en4 a17
Effectivity_End_Date en0 m entity en4 a18
Unit_of_Measure en0 m 1 4 STRING a19
Engineering_Change_Notice en0 m 1 7 INTEGER a20
Change_Effected_by en0 m 1 20 STRING a21
Date_of_Change en0 m entity en4 a22
Phases_out_Part_Number en0 m 1 15 STRING a23
Phased_out_by_Part_Number en0 m 1 15 STRING a24
Number_of_Levels en0 m 1 2 INTEGER a25
Number_of_components en0 m 1 3 INTEGER a26

PROCESS PLAN

has_BOM_CHILD e15 m entity en5 a27
has_MFG_OPERATION_ASSIGNMENT e15 m entity en6 a28
has_MFG_CELL_CONFIGURATION e15 m entity en7 a29

Process_plan_ID e15 m 1 7 INTEGER a30
Process_Description e15 m 1 60 STRING a31
Part_number en5 m 1 15 STRING a33

```
has_ENGINEERING_RESOURCE en5 m LIST 1 # en8 a32

Number_of_components en5 m 1 3 INTEGER a34
Part_Type en5 m enumeration t2 a35
Quantity_per_Assembly en5 m REAL a36
Effectivity_Start_Date en5 m entity en4 a37
Effectivity_End_Date en5 m entity en4 a38
Unit_of_Measure en5 m 1 4 STRING a39
Lead_Time_Offset en5 m 1 9 INTEGER a40
Engineering_Change_Notice en5 m 1 7 INTEGER a41
Change_Effected_by en5 m 1 20 STRING a42
Change_Date en5 m entity en4 a43
Phases_Out_Part_Number en5 m 1 15 STRING a44
Phased_Out_by_Part_Number en5 m 1 15 STRING a45
```

BOM-CHILD

```
has_RESOURCE_ASSIGNMENT en6 m LIST 1 # en9 a46
has_MFG_FACILITY_ASSIGNMENT en6 m LIST 1 # en10 a47

Mfg_Operation_ID en6 m 1 7 INTEGER a48
Mfg_Operation_Description en6 m 1 60 STRING a49
Preceed_Mfg_Operation_ID en6 m 1 7 INTEGER a50
Next_Mfg_Operation_ID en6 m 1 7 INTEGER a51
Alternate_Mfg_Operation_ID en6 m 1 7 INTEGER a52
Setup_time_per_item en6 m REAL a53
Machining_time_per_item en6 m REAL a54
Handling_time_per_item en6 m REAL a55
Operation_time_per_item en6 m REAL a56
Scrap_rate en6 m REAL a57
```

MANUFACTURING OPERATION ASSIGNMENT

```
has_RESOURCE en9 m entity en11 a58

Resource_Type en9 m enumeration t3 a59
Resource_Type en9 m entity t3 a60
Quantity_Required en9 m REAL a61
Unit_of_Measure en9 m 1 4 STRING a62
```

RESOURCE ASSIGNMENT

```
has_MANUFACTURING_FACILITY en10 m entity en12 a63
Feed en10 m 1 4 INTEGER a64
Speed en10 m 1 6 INTEGER a65
Depth_of_cut en10 m 1 3 INTEGER a66
Number_of_passes en10 m 1 5 INTEGER a67
Remarks en10 m 1 60 STRING a68
```

MANUFACTURING FACILITY ASSIGNMENT

```
has_MANUFACTURING_FACILITY en7 m LIST 1 # en12 a69

Mfg_Cell_Group_ID en7 m 1 2 INTEGER a70
Number_of_Mfg_Stations en7 m 1 2 INTEGER a71
Mfg_station_1 en7 m 1 7 INTEGER a72
Description_station_1 en7 m 1 60 STRING a73
Mfg_station_2 en7 m 1 7 INTEGER a74
Description_station_2 en7 m 1 60 STRING a75
Mfg_station_3 en7 m 1 7 INTEGER a76
Description_station_3 en7 m 1 60 STRING a77
Mfg_station_4 en7 m 1 7 INTEGER a78
Description_station_4 en7 m 1 60 STRING a79
Mfg_station_5 en7 m 1 7 INTEGER a80
Description_station_5 en7 m 1 60 STRING a81
```

MANUFACTURING CELL CONFIGURATION

```
Resource_ID en11 m 1 15 STRING a82
Resource_Type en11 m enumeration t3 a83
Description en11 m 1 60 STRING a84
Location en11 m 1 15 STRING a85
Account_Number en11 m 1 15 INTEGER a86
Unit_of_Measure en11 m 1 4 STRING a87
Unit_Price en11 m REAL a88
Buy_Make_Supply_Code en11 m enumeration t5 a89
Catalogue_Order_Number en11 m 1 30 STRING a90
Purchasing_Lead_Time en11 m 1 7 INTEGER a91
Last_Order_Date en11 m entity en4 a92
Quantity_Ordered en11 m 1 9 INTEGER a93
Effectivity_Start_Date en11 m entity en4 a94
Effectivity_End_Date en11 m entity en4 a95
Stock_on_hand en11 m 1 9 INTEGER a96
Allocated_Reserved_Stock en11 m 1 9 INTEGER a97
Scrap_Value en11 m REAL a98
Scrap_unit_of_Measure en11 m 1 4 STRING a99
```

MANUFACTURING RESOURCE

```
has_PROCESS_PLAN en8 m entity e15 a100

Engineering_Resource en8 m 1 10 STRING a101
Location en8 m 1 10 STRING a102
```

ENGINEERING RESOURCE

```
Customer_ID en1 m 1 7 INTEGER a103
Company_Name en1 m 1 40 STRING a104
Address en1 m 1 60 STRING a105
Contact_Person en1 m 1 25 STRING a106
Telephone en1 m 1 20 STRING a107
Fax en1 m 1 20 STRING a108
```

CUSTOMERS

```
has_SHOP_FLOOR_STATUS en2 m entity en3 a109
has_BOM_CHILD en2 m entity en5 a110

Priority en2 m 1 3 INTEGER a111
Order_Status en2 m enumeration t4 a112
Order_Status en2 m entity t4 a113
Planned_Quantity en2 m 1 9 INTEGER a114
Unit_of_Measure en2 m 1 4 STRING a115
Schedule_Start_Date en2 m entity en4 a116
Schedule_End_Date en2 m entity en4 a117
```

SCHEDULE

```
has_BOM_CHILD en3 m entity en5 a118

Actual_Quantity_Produced en3 m 1 9 INTEGER a119
Station_Utilisation_Rate en3 m REAL a120
Actual_Capacity_Utilised en3 m REAL a121
```

SHOP FLOOR STATUS

```
Asset_ID en12 m 1 7 INTEGER a122
Description en12 m 1 60 STRING a123
Location en12 m 1 15 STRING a124
Working_Capacity en12 m 1 3 INTEGER a125
Labor_Cost_per_hour en12 m REAL a126
Handling_Cost_per_hour en12 m REAL a127
```

MANUFACTURING FACILITY

```
has_RESOURCE e44 m LIST 1 # en11 a128

Supplier_ID e44 m 1 7 INTEGER a129
Company_Name e44 m 1 40 STRING a130
Address e44 m 1 60 STRING a131
Contact_Person e44 m 1 25 STRING a132
Telephone e44 m 1 20 STRING a133
Fax e44 m 1 20 STRING a134
```

SUPPLIERS

```
Personnel_ID e46 m 1 15 STRING a135
Name e46 m 1 30 STRING a136
Address e46 m 1 60 STRING a137
Telephone e46 m 1 20 STRING a138
Salary e46 m REAL a139
Skill e46 m 1 30 STRING a140
Skill_level e46 m 1 2 INTEGER a141
Remarks e46 m 1 60 STRING a142
```

PERSONNEL

```
has_SUPPLIER e47 m entity e44 a143

Last_Service_Maintenance_Date e47 m entity en4 a144
Repaired_on e47 m entity en4 a145
Repair_Work_Order_Number e47 m 1 7 INTEGER a146
Max_job_size_X_axis e47 m REAL a147
Max_job_size_Y_axis e47 m REAL a148
Max_job_size_Z_axis e47 m REAL a149
Accuracy e47 m REAL a150
Machining_Cost_per_hour e47 m REAL a151
Horse_Power e47 m 1 7 INTEGER a152
Speed_Range_Min e47 m 1 6 INTEGER a153
Speed_Range_Max e47 m 1 6 INTEGER a154
Feed_Range_Min e47 m 1 4 INTEGER a155
Feed_Range_Max e47 m 1 4 INTEGER a156
Payload e47 m 1 5 INTEGER a157
Working_Envelope_X_axis e47 m REAL a158
Working_Envelope_Y_axis e47 m REAL a159
Working_Envelope_Z_axis e47 m REAL a160
Working_Envelope_A_axis e47 m REAL a161
Working_Envelope_B_axis e47 m REAL a162
Setup_Time e47 m REAL a163
Tool_Change_Time e47 m REAL a164
Feed_Change_Time e47 m REAL a165
Table_Rotation_Time e47 m REAL a166
Tool_Adjustment_Time e47 m REAL a167
Rapid_Tranverse_Rate e47 m REAL a168
```

MACHINING
FACILITY

```
day en4 m 1 2 INTEGER a169
month en4 m 1 2 INTEGER a170
year en4 m 1 2 INTEGER a171
```

DATE

4. EXPRESS to SQL Compiler generated datafile

```
create table index_table (
sys_name char(10),
entity_name char(80)
);

create table data_dict (
attribute_name char(80),
table_name char(10),
mand_opt char(1),
class char(10),
index_1 char(10),
index_2 char(10),
type char(10),
attribute_token char(80)
);

create table logical
(
logical_id number(10),
logical_type char(10)
);

insert into logical (logical_id, logical_type)
values (0, 'FALSE'
);

insert into logical (logical_id, logical_type)
values (1, 'TRUE'
);

insert into logical (logical_id, logical_type)
values (2, 'UNKNOWN'
);

create table types
(
type_name char(80),
opt char(80),
class char(15)
);

create table id_table
(
table_name char(80),
column_name char(80)
);

create table e2
(
e1_id number(10),
en0_id number(10)
);

create table e3
(
e1_id number(10),
en1_id number(10)
);

create table e4
(
e1_id number(10),
en2_id number(10),
e4_id number(10)
);
```

```
create table e4_ps
(
e4_ps_pred number(10),
e4_ps_succ number(10)
);

create table e5
(
e1_id number(10),
en3_id number(10),
e5_id number(10)
);

create table e5_ps
(
e5_ps_pred number(10),
e5_ps_succ number(10)
);

create table e6
(
e1_id number(10),
en4_id number(10)
);

create table e7
(
e1_id number(10),
en4_id number(10)
);

create table e8
(
e1_id number(10),
t1_id number(10)
);

create table e9
(
e1_id number(10),
t1_id number(10)
);

create table e10
(
e1_id number(10),
en4_id number(10)
);

create table e1
(
e1_id number(10),
e1_Mfg_Order_Number number(7),
e1_Preceeding_Order_Number number(7),
e1_Description char(60),
e1_Unit_of_Measure char(4),
e1_Unit_Price number(7,2),
e1_Order_Quantity number(9)
);

create table e11
(
en0_id number(10),
en5_id number(10),
e11_id number(10)
);
```

```
create table e11_ps
(
e11_ps_pred number(10),
e11_ps_succ number(10)
);

create table e12
(
en0_id number(10),
en4_id number(10)
);

create table e13
(
en0_id number(10),
en4_id number(10)
);

create table e14
(
en0_id number(10),
en4_id number(10)
);

create table en0
(
en0_id number(10),
en0_Parent_part_number char(15),
en0_Unit_of_Measure char(4),
en0_Engineering_Change_Notice number(7),
en0_Change_Effected_by char(20),
en0_Phases_out_Part_Number char(15),
en0_Phased_out_by_Part_Number char(15),
en0_Number_of_Levels number(2),
en0_Number_of_components number(3)
);

create table e16
(
e15_id number(10),
en5_id number(10)
);

create table e17
(
e15_id number(10),
en6_id number(10)
);

create table e18
(
e15_id number(10),
en7_id number(10)
);

create table e15
(
e15_id number(10),
e15_Process_plan_ID number(7),
e15_Process_Description char(60)
);

create table e19
(
en5_id number(10),
en8_id number(10),
e19_id number(10)
);
```

```
create table e19_ps
(
e19_ps_pred number(10),
e19_ps_succ number(10)
);

create table e20
(
en5_id number(10),
t2_id number(10)
);

create table e21
(
en5_id number(10),
en4_id number(10)
);

create table e22
(
en5_id number(10),
en4_id number(10)
);

create table e23
(
en5_id number(10),
en4_id number(10)
);

create table en5
(
en5_id number(10),
en5_Part_number char(15),
en5_Number_of_components number(3),
en5_Quantity_per_Assembly number(7,2),
en5_Unit_of_Measure char(4),
en5_Lead_Time_Offset number(9),
en5_Engineering_Change_Notice number(7),
en5_Change_Effected_by char(20),
en5_Phases_Out_Part_Number char(15),
en5_Phased_Out_by_Part_Number char(15)
);

create table e24
(
en6_id number(10),
en9_id number(10),
e24_id number(10)
);

create table e24_ps
(
e24_ps_pred number(10),
e24_ps_succ number(10)
);

create table e25
(
en6_id number(10),
en10_id number(10),
e25_id number(10)
);

create table e25_ps
(
e25_ps_pred number(10),
e25_ps_succ number(10)
);
```

```
create table en6
(
en6_id number(10),
en6_Mfg_Operation_ID number(7),
en6_Mfg_Operation_Description char(60),
en6_Preceed_Mfg_Operation_ID number(7),
en6_Next_Mfg_Operation_ID number(7),
en6_Alternate_Mfg_Operation_ID number(7),
en6_Setup_time_per_item number(7,2),
en6_Machining_time_per_item number(7,2),
en6_Handling_time_per_item number(7,2),
en6_Operation_time_per_item number(7,2),
en6_Scrap_rate number(7,2)
);

create table e26
(
en9_id number(10),
en11_id number(10)
);

create table e27
(
en9_id number(10),
t3_id number(10)
);

create table e28
(
en9_id number(10),
t3_id number(10)
);

create table en9
(
en9_id number(10),
en9_Quantity_Required number(7,2),
en9_Unit_of_Measure char(4)
);

create table e29
(
en10_id number(10),
en12_id number(10)
);

create table en10
(
en10_id number(10),
en10_Feed number(4),
en10_Speed number(6),
en10_Depth_of_cut number(3),
en10_Number_of_passes number(5),
en10_Remarks char(60)
);

create table e30
(
en7_id number(10),
en12_id number(10),
e30_id number(10)
);

create table e30_ps
(
e30_ps_pred number(10),
e30_ps_succ number(10)
);
```

```
create table en7
(
en7_id number(10),
en7_Mfg_Cell_Group_ID number(2),
en7_Number_of_Mfg_Stations number(2),
en7_Mfg_station_1 number(7),
en7_Description_station_1 char(60),
en7_Mfg_station_2 number(7),
en7_Description_station_2 char(60),
en7_Mfg_station_3 number(7),
en7_Description_station_3 char(60),
en7_Mfg_station_4 number(7),
en7_Description_station_4 char(60),
en7_Mfg_station_5 number(7),
en7_Description_station_5 char(60)
);

create table e31
(
en11_id number(10),
t3_id number(10)
);

create table e32
(
en11_id number(10),
t5_id number(10)
);

create table e33
(
en11_id number(10),
en4_id number(10)
);

create table e34
(
en11_id number(10),
en4_id number(10)
);

create table e35
(
en11_id number(10),
en4_id number(10)
);

create table en11
(
en11_id number(10),
en11_Resource_ID char(15),
en11_Description char(60),
en11_Location char(15),
en11_Account_Number number(15),
en11_Unit_of_Measure char(4),
en11_Unit_Price number(7,2),
en11_Catalogue_Order_Number char(30),
en11_Purchasing_Lead_Time number(7),
en11_Quantity_Ordered number(9),
en11_Stock_on_hand number(9),
en11_Allocated_Reserved_Stock number(9),
en11_Scrap_Value number(7,2),
en11_Scrap_unit_of_Measure char(4)
);
```

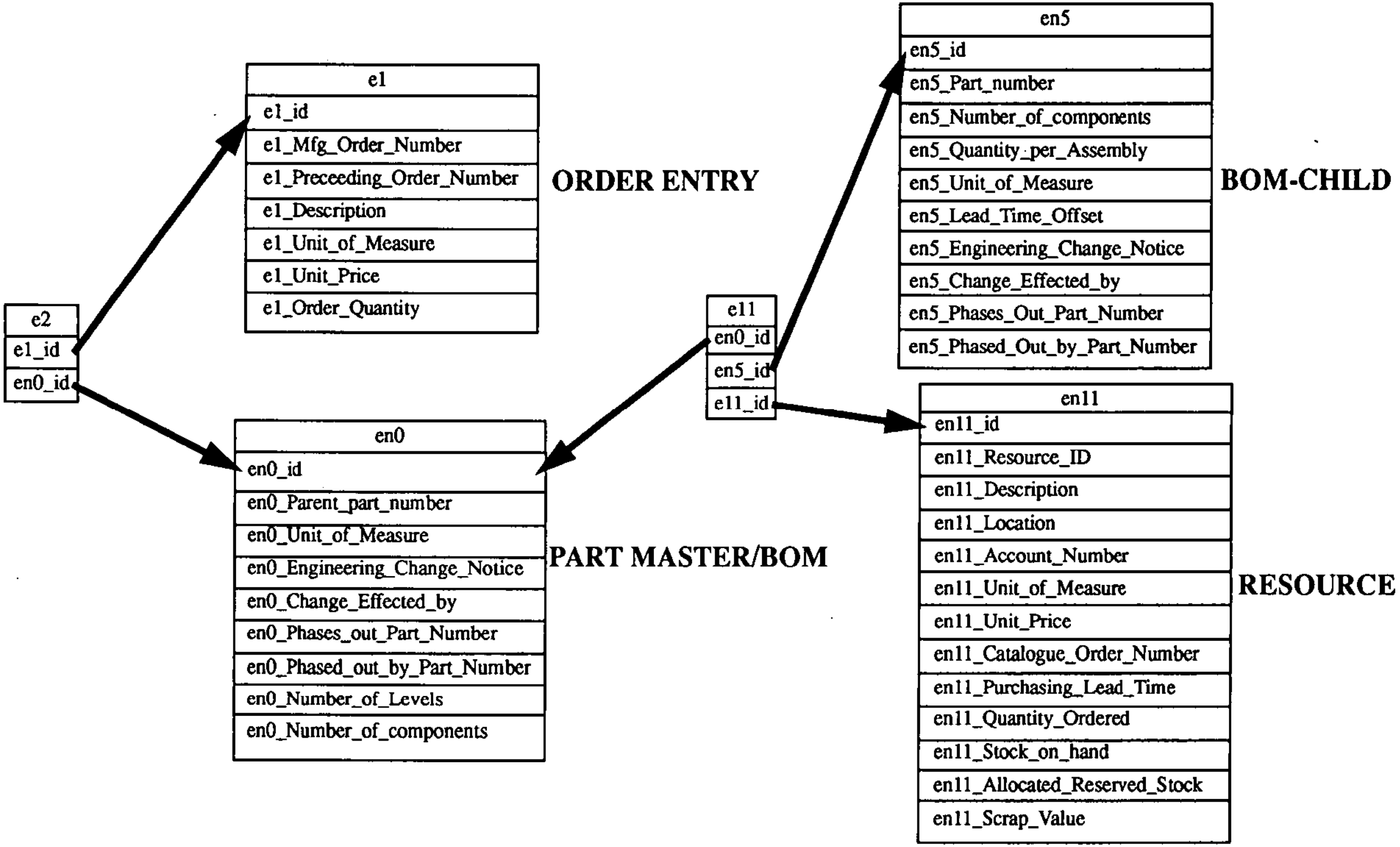
```
create table e36
(
en8_id number(10),
e15_id number(10)
);

create table en8
(
en8_id number(10),
en8_Engineering_Resource char(10),
en8_Location char(10)
);

create table en1
(
en1_id number(10),
en1_Customer_ID number(7),
en1_Company_Name char(40),
en1_Address char(60),
en1_Contact_Person char(25),
en1_Telephone char(20),
en1_Fax char(20)
);

create table e37
(
en2_id number(10),
en3_id number(10)
);

create table e38
(
en2_id number(10),
en5_id number(10)
);

create table e39
(
en2_id number(10),
t4_id number(10)
);

create table e40
(
en2_id number(10),
t4_id number(10)
);

create table e41
(
en2_id number(10),
en4_id number(10)
);

create table e42
(
en2_id number(10),
en4_id number(10)
);

create table en2
(
en2_id number(10),
en2_Priority number(3),
en2_Planned_Quantity number(9),
en2_Unit_of_Measure char(4)
);
```

```
create table e43
(
en3_id number(10),
en5_id number(10)
);

create table en3
(
en3_id number(10),
en3_Actual_Quantity_Produced number(9),
en3_Station_Utilisation_Rate number(7,2),
en3_Actual_Capacity_Utilised number(7,2)
);

create table en12
(
en12_id number(10),
en12_Asset_ID number(7),
en12_Description char(60),
en12_Location char(15),
en12_Working_Capacity number(3),
en12_Labor_Cost_per_hour number(7,2),
en12_Handling_Cost_per_hour number(7,2)
);

create table e45
(
e44_id number(10),
en11_id number(10),
e45_id number(10)
);

create table e45_ps
(
e45_ps_pred number(10),
e45_ps_succ number(10)
);

create table e44
(
e44_id number(10),
e44_Supplier_ID number(7),
e44_Company_Name char(40),
e44_Address char(60),
e44_Contact_Person char(25),
e44_Telephone char(20),
e44_Fax char(20)
);

create table e46
(
e46_id number(10),
en12_id number(10),
e46_Personnel_ID char(15),
e46_Name char(30),
e46_Address char(60),
e46_Telephone char(20),
e46_Salary number(7,2),
e46_Skill char(30),
e46_Skill_level number(2),
e46_Remarks char(60)
);

create table e48
(
e47_id number(10),
e44_id number(10)
);
```

```
create table e49
(
e47_id number(10),
en4_id number(10)
);

create table e50
(
e47_id number(10),
en4_id number(10)
);

create table e47
(
e47_id number(10),
en12_id number(10),
e47_Repair_Work_Order_Number number(7),
e47_Max_job_size_X_axis number(7,2),
e47_Max_job_size_Y_axis number(7,2),
e47_Max_job_size_Z_axis number(7,2),
e47_Accuracy number(7,2),
e47_Machining_Cost_per_hour number(7,2),
e47_Horse_Power number(7),
e47_Speed_Range_Min number(6),
e47_Speed_Range_Max number(6),
e47_Feed_Range_Min number(4),
e47_Feed_Range_Max number(4),
e47_Payload number(5),
e47_Working_Envelope_X_axis number(7,2),
e47_Working_Envelope_Y_axis number(7,2),
e47_Working_Envelope_Z_axis number(7,2),
e47_Working_Envelope_A_axis number(7,2),
e47_Working_Envelope_B_axis number(7,2),
e47_Setup_Time number(7,2),
e47_Tool_Change_Time number(7,2),
e47_Feed_Change_Time number(7,2),
e47_Table_Rotation_Time number(7,2),
e47_Tool_Adjustment_Time number(7,2),
e47_Rapid_Tranverse_Rate number(7,2)
);

create table en4
(
en4_id number(10),
en4_day number(2),
en4_month number(2),
en4_year number(2)
);

create table t1
(
t1_id number(10),
t1_name char(12)
);

create table t2
(
t2_id number(10),
t2_name char(16)
);

create table t3
(
t3_id number(10),
t3_name char(19)
);
```

```
create table t4
(
t4_id number(10),
t4_name char(20)
);

create table t5
(
t5_id number(10),
t5_name char(6)
);
```

5. Example of relational database tables relationship



e2
e1_id
en0_id

ORDER ENTRY

e1
e1_id
e1_Mfg_Order_Number
e1_Preceeding_Order_Number
e1_Description
e1_Unit_of_Measure
e1_Unit_Price
e1_Order_Quantity

PART MASTER/BOM

en0
en0_id
en0_Parent_part_number
en0_Unit_of_Measure
en0_Engineering_Change_Notice
en0_Change_Effected_by
en0_Phases_out_Part_Number
en0_Phased_out_by_Part_Number
en0_Number_of_Levels
en0_Number_of_components

e11
en0_id
en5_id
e11_id

BOM-CHILD

en5
en5_id
en5_Part_number
en5_Number_of_components
en5_Quantity_per_Assembly
en5_Unit_of_Measure
en5_Lead_Time_Offset
en5_Engineering_Change_Notice
en5_Change_Effected_by
en5_Phases_Out_Part_Number
en5_Phased_Out_by_Part_Number

RESOURCE

en11
en11_id
en11_Resource_ID
en11_Description
en11_Location
en11_Account_Number
en11_Unit_of_Measure
en11_Unit_Price
en11_Catalogue_Order_Number
en11_Purchasing_Lead_Time
en11_Quantity_Ordered
en11_Stock_on_hand
en11_Allocated_Reserved_Stock
en11_Scrap_Value

APPENDIX XV

Report on
IDEF$_{1X}$ Entity-Attribute relationship model
(to be translated to EXPRESS schema)

{ENTITY, IE1, ORDER ENTRY, {MFG ORDER NUMBER,}, {Preceeding Mfg Order Number, Description, Product Effectivity Start Date, Product Effectivity End Date, Type, Due Date,Unit of Measure,Unit Price, Order Quantity}}

{ENTITY, IE2, PART MASTER-BOM, {PARENT PART NUMBER,}, {Effectivity Start Date, Effectivity End Date, Unit of Measure,Engineering Change Notice,Change Effected by,Date of Change,Phases out Part Number, Phased out by Part Number, Number of Levels, Number of components (children)}}

{ENTITY, IE4, PROCESS PLAN, {PROCESS PLAN ID,}, {Process Description,}}

{DEPENDENT ENTITY, DE21, BOM CHILD, {PART NUMBER,}, {Number of components (children),Part Type, Quantity per Assembly,Effectivity Start Date,Effectivity End Date,Unit of Measure,Lead Time Offset, Engineering Change Notice, Change Effected by,Change Date,Phases Out Part Number,Phased Out by Part Number}}

{DEPENDENT ENTITY, DE41 , MFG OPERATION ASSIGNMENT, {MFG OPERATION ID,}, {Mfg Operation Description, Preceeding Mfg Operation ID,Next Mfg Operation ID, Alternative Mfg Operation ID, Setup time per item,Machining time per item, Handling time per item,Operation time per item, Scrap rate,, }}

{DEPENDENT ENTITY, DE43, RESOURCE ASSIGNMENT, {,}, {Resource Type,Quantity Required,Unit of Measure}}

{DEPENDENT ENTITY, DE42, MFG FACILITY ASSIGNMENT, {}, {Feed,Speed, Depth of cut,Number of passes, Remarks}}

{ENTITY, IE5, MFG CELL CONFIGURATION, {MFG CELL GROUP ID,}, {Number of Mfg Stations}}

{ENTITY, IE6, RESOURCE, {RESOURCE ID,}, {Resource Type, Description, Location, Account Number, Unit of Measure, Unit Price,Buy/Make/Supply Code,Catalogue Order Number,Purchasing Lead Time, Last Order Date, Quantity Ordered,Effectivity Start Date,Effectivity End Date,Stock on-hand, Allocated/Reserved Stock,Scrap Value, Unit of Measure for Scrap}}

{ENTITY, IE3, ENGINEERING RESOURCE, {}, {Engineering Resource,Location}}

{ENTITY, IE7, CUSTOMER, {CUSTOMER ID}, {Company/Name, Address, Contact Person,Telephone,Fax}}

{ENTITY, IE8, SCHEDULE, {}, {Priority,Order Status,Planned Quantity,Unit of Measure, Schedule Start Date, Schedule End Date}}

{ENTITY, IE9, SHOP FLOOR STATUS, {}, {Actual Quantity Produced,Work Centre/Cell Utilisation Rate, Actual Capacity Utilised}}

{ENTITY, IE10, MANUFACTURING FACILITY, {ASSET ID,}, {Description,Location,Working Capacity, Labor Cost per hour,Handling Cost per hour,,}}

{ENTITY, IE11, SUPPLIER, {SUPPLIER ID}, {Company/Name, Address, Contact Person, Telephone,Fax}}

{DEPENDENT ENTITY, DE101, PERSONNEL, { }, {Personnel ID,Name,Address,Telephone,Salary,Skill,Skill level,Remarks}}

{DEPENDENT ENTITY, DE102, MACHINE, {}, {Last Service/Maintenance Date,Repaired on, Repair Work Order Number,Max. job size accommodated - X/Y/Z axes,Accuracy,Machining Cost per hour, Horse Power,Speed Range (Max./Min.),Feed Range,Payload,Working Envelope - X/Y/Z/A/B axes,Setup Time, Tool Change Time,Feed Change Time,Table Rotation Time,Tool Adjustment Time,Rapid Tranverse Rate}}

```
{CATEGORIZATION, CR1, , COMPLETE, {IE10}, {DE101,DE102}

{RELATION, RL1, , NON-SPECIFIC, IE2, DE21, OM}

{RELATION, RL2, , NON-SPECIFIC, IE4, DE21, O}

{RELATION, RL3, , NON-SPECIFIC, DE41 , DE43, OM}

{RELATION, RL4, , NON-SPECIFIC, IE1, IE2, O}

{RELATION, RL5, , NON-SPECIFIC, IE4, DE41 , O}

{RELATION, RL6, , NON-SPECIFIC, DE41 , DE42, OM}

{RELATION, RL7, , NON-SPECIFIC, DE43, IE6, O}

{RELATION, RL8, , NON-SPECIFIC, IE1, IE7, O}

{RELATION, RL9, , NON-SPECIFIC, IE8, IE9, O}

{RELATION, RL10, , NON-SPECIFIC, IE9, DE21, O}

{RELATION, RL11, , NON-SPECIFIC, IE8, DE21, O}

{RELATION, RL12, , NON-SPECIFIC, IE4, IE5, ZOM}

{RELATION, RL13, , NON-SPECIFIC, IE3, IE4, O}

{RELATION, RL14, , NON-SPECIFIC, IE1, IE8, OM}

{RELATION, RL15, , NON-SPECIFIC, IE1, IE9, OM}

{RELATION, RL16, , NON-SPECIFIC, IE5, IE10, OM}

{RELATION, RL17, , NON-SPECIFIC, IE11, IE6, OM}

{RELATION, RL18, , NON-SPECIFIC, DE102, IE11, O}

{RELATION, RL19, , NON-SPECIFIC, DE21, IE3, OM}

{RELATION, RL20, , NON-SPECIFIC, DE42, IE10, O}

Done.
```

APPENDIX XVI

Report on
$IDEF_0$ MCS function model

[Diagram: A-0]

Activity: [A0] Part manufacture with MCS

Arrow: Product Order
Input From: Product Order
Input To: [A0] Part manufacture with MCS

Arrow: Shop Floor Status Report
Input From: Shop Floor Status Report
Input To: [A0] Part manufacture with MCS

Arrow: Customer Order Enquiry & Request
Control From: Customer Order Enquiry & Request
Control To: [A0] Part manufacture with MCS

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A0] Part manufacture with MCS

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A0] Part manufacture with MCS

Arrow: Engineering Resources
Output From: [A0] Part manufacture with MCS
Output To: Engineering Resources

Arrow: Schedules
Output From: [A0] Part manufacture with MCS
Output To: Schedules

Arrow: Process Plans
Output From: [A0] Part manufacture with MCS
Output To: Process Plans

Arrow: Shop Floor Status
Output From: [A0] Part manufacture with MCS
Output To: Shop Floor Status

Arrow: Order Acknowledgement
Output From: [A0] Part manufacture with MCS
Output To: Order Acknowledgement

Arrow: Finished Products
Output From: [A0] Part manufacture with MCS
Output To: Finished Products

Arrow: Rejects/Scrap
Output From: [A0] Part manufacture with MCS
Output To: Rejects/Scrap

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A0] Part manufacture with MCS
Arrow: Manufacturing Cell
Mechanism From: Manufacturing Cell
Mechanism To: [A0] Part manufacture with MCS

Arrow: Procurement of Manufacturing Resources
Output From: [A0] Part manufacture with MCS
Output To: Procurement of Manufacturing Resources

Arrow: Tansaction Report for Progress/Delays & Resources Shortfall
Output From: [A0] Part manufacture with MCS
Output To: Tansaction Report for Progress/Delays & Resources Shortfall

Arrow: Allocation of Manufacturing Resources & Facilities/Picklist
Output From: [A0] Part manufacture with MCS
Output To: Allocation of Manufacturing Resources & Facilities/Picklist

Arrow: Production Report
Input From: Production Report
Input To: [A0] Part manufacture with MCS

Arrow: Manufacturing Resources & Facilities Requisition
Input From: Manufacturing Resources & Facilities Requisition
Input To: [A0] Part manufacture with MCS

[Diagram: A0] Part manufacture with MCS

Activity: [A1] Pre-Planning

Activity: [A2] Planning for Manufacture

Activity: [A3] Manufacturing Control

Arrow: Customer Order Enquiry & Request
Control From: Customer Order Enquiry & Request
Control To: [A1] Pre-Planning

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A1] Pre-Planning

Arrow: Confirmed Orders
Output From: [A1] Pre-Planning
Input To: [A2] Planning for Manufacture

Arrow: Order Acknowledgement
Output From: [A1] Pre-Planning
Output To: Order Acknowledgement

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A2] Planning for Manufacture

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A3] Manufacturing Control

Arrow: Finished Products
Output From: [A3] Manufacturing Control
Output To: Finished Products

Arrow: Rejects/Scrap
Output From: [A3] Manufacturing Control
Output To: Rejects/Scrap

Arrow: Shop Floor Status Report
Output From: [A3] Manufacturing Control
Output To: Shop Floor Status

Arrow: Product Order
Input From: Product Order
Input To: [A1] Pre-Planning

Arrow: Shop Floor Data Acquisition
Mechanism From: Shop Floor Data Acquisition
Mechanism To: [A3] Manufacturing Control

Arrow:
Output From: [A2] Planning for Manufacture
Output To: Process Plans

Arrow:
Output From: [A2] Planning for Manufacture
Output To: Schedules

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A1] Pre-Planning

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A2] Planning for Manufacture

Arrow: Procurement of Manufacturing Resources
Output From: [A2] Planning for Manufacture
Output To: Procurement of Manufacturing Resources

Arrow: Production Report
Output From: [A2] Planning for Manufacture
Output To: Tansaction Report for Progress/Delays & Resources Shortfall

Arrow:
Output From: [A2] Planning for Manufacture
Output To: Allocation of Manufacturing Resources & Facilities/Picklist

Arrow:
Output From: [A2] Planning for Manufacture
Output To: Engineering Resources

Arrow: Schedules
Output From: [A2] Planning for Manufacture
Input To: [A3] Manufacturing Control

Arrow: Process Plans
Output From: [A2] Planning for Manufacture
Input To: [A3] Manufacturing Control

Arrow: Allocation of Manufacturing Resources & Facilities/Picklist
Output From: [A2] Planning for Manufacture
Input To: [A3] Manufacturing Control

Arrow: Engineering Resources
Output From: [A2] Planning for Manufacture
Input To: [A3] Manufacturing Control

Arrow: Manufacturing Cell
Mechanism From: Manufacturing Cell
Mechanism To: [A3] Manufacturing Control

Arrow: Shop Floor Status Report
Output From: [A3] Manufacturing Control
Input To: [A2] Planning for Manufacture

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A2] Planning for Manufacture

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A1] Pre-Planning

Arrow: Production Report
Output From: [A2] Planning for Manufacture
Output To: [A2] Planning for Manufacture

Arrow: Manufacturing Resources & Facilities Requisition
Input From: Manufacturing Resources & Facilities Requisition
Input To: [A2] Planning for Manufacture

[Diagram: A2] Planning for Manufacture

Activity: [A21] Production Planning

Activity: [A23] Process Planning

Activity: [A24] Product Design (CAD/CAM)

Activity: [A22] Finite Capacity Scheduler

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A23] Process Planning

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A24] Product Design (CAD/CAM)

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A22] Finite Capacity Scheduler

Arrow: Confirmed Orders
Input From: Confirmed Orders
Input To: [A21] Production Planning

Arrow: Process Plans
Output From: [A23] Process Planning
Output To: Process Plans

Arrow: Allocation of Manufacturing Resources & Facilities/Picklist
Output From: [A21] Production Planning
Output To: Allocation of Manufacturing Resources & Facilities/Picklist

Arrow: Procurement of Manufacturing Resources
Output From: [A21] Production Planning
Output To: Procurement of Manufacturing Resources

Arrow: Production Report
Input From: Production Report
Input To: [A21] Production Planning

Arrow: Process Plans
Output From: [A21] Production Planning
Output To: Process Plans

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A23] Process Planning

Arrow: Schedules
Output From: [A22] Finite Capacity Scheduler
Output To: Schedules

Arrow: BOM
Output From: [A24] Product Design (CAD/CAM)
Output To: BOM

Arrow: Engineering Resources
Output From: [A24] Product Design (CAD/CAM)
Output To: Engineering Resources

Arrow: BOM
Output From: [A21] Production Planning
Output To: BOM

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A21] Production Planning

Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A24] Product Design (CAD/CAM)

Arrow: Confirmed Orders
Input From: Confirmed Orders
Input To: [A23] Process Planning

Arrow: Confirmed Orders
Input From: Confirmed Orders
Input To: [A24] Product Design (CAD/CAM)

Arrow: Manufacturing Resources & Facilities Requisition
Output From: [A23] Process Planning
Input To: [A21] Production Planning

Arrow: Manufacturing Resources & Facilities Requisition
Output From: [A24] Product Design (CAD/CAM)
Input To: [A21] Production Planning

Arrow: Transaction Report for Progress/Delays & Resources Shortfall
Output From: [A21] Production Planning
Output To: Transaction Report for Progress/Delays & Resources Shortfall

Arrow: Shop Floor Status Report
Input From: Shop Floor Status Report
Input To: [A21] Production Planning

Arrow: Shop Floor Status Report
Input From: Shop Floor Status Report
Input To: [A22] Finite Capacity Scheduler

Arrow: Manufacturing Orders (Unscheduled)
Output From: [A21] Production Planning
Input To: [A22] Finite Capacity Scheduler

Arrow: MCS Functions
Mechanism From: MCS Functions
Mechanism To: [A21] Production Planning

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A21] Production Planning

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A23] Process Planning

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A22] Finite Capacity Scheduler

Arrow: Enterprise Manufacturing Capability
Control From: Enterprise Manufacturing Capability
Control To: [A24] Product Design (CAD/CAM)

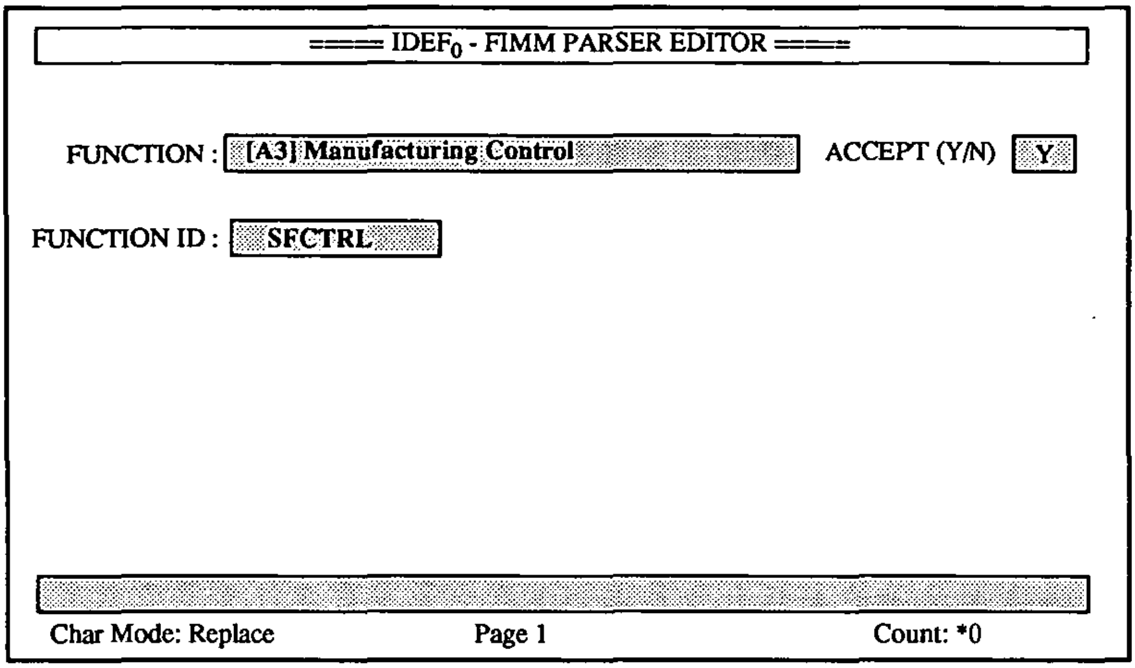
Arrow: Availability of Manufacturing Resources & Facilities
Control From: Availability of Manufacturing Resources & Facilities
Control To: [A22] Finite Capacity Scheduler

Done.

APPENDIX XVII

$\text{IDEF}_{0/1X}$ Parser user interfaces

(A) User interface for selection of functions defined in IDEF$_0$ functional model



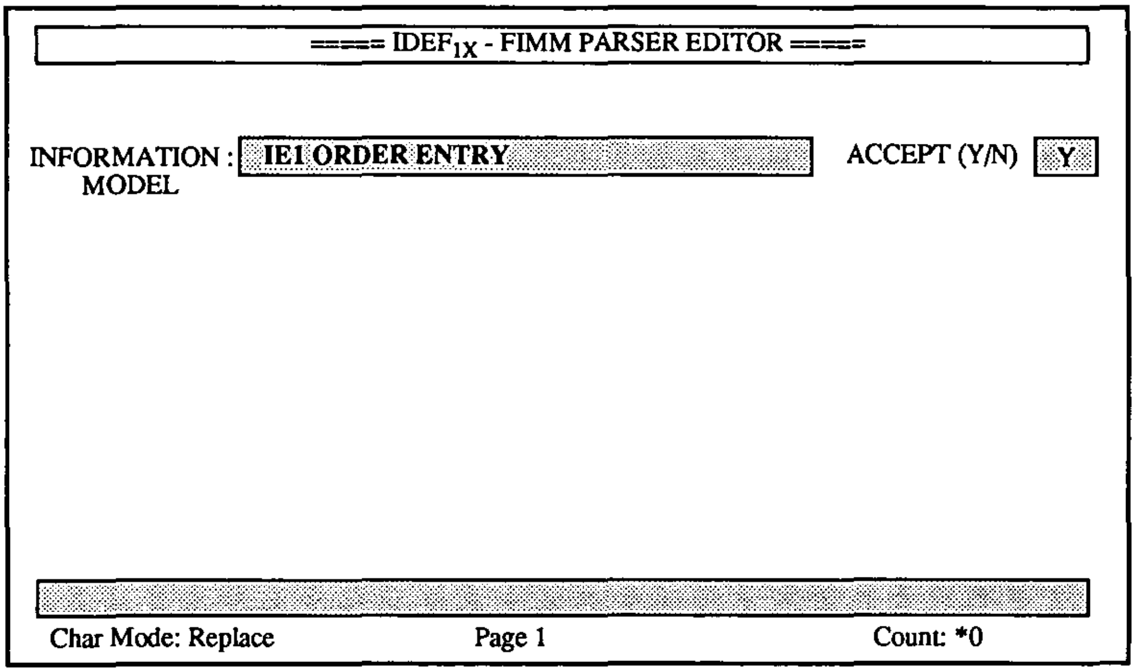
==== IDEF$_0$ - FIMM PARSER EDITOR ====

FUNCTION : [A3] Manufacturing Control ACCEPT (Y/N) Y

FUNCTION ID : SFCTRL

Char Mode: Replace Page 1 Count: *0

(B) User interface for selection of information entities defined in IDEF$_{1X}$ information model



==== IDEF$_{1X}$ - FIMM PARSER EDITOR ====

INFORMATION MODEL : IE1 ORDER ENTRY ACCEPT (Y/N) Y

Char Mode: Replace Page 1 Count: *0

APPENDIX XVIII

Program listings for IDEF$_{0/1X}$ Parser

1. Program structure for IDEF$_{0/1X}$ Parser

2. Program listing for idef0_act

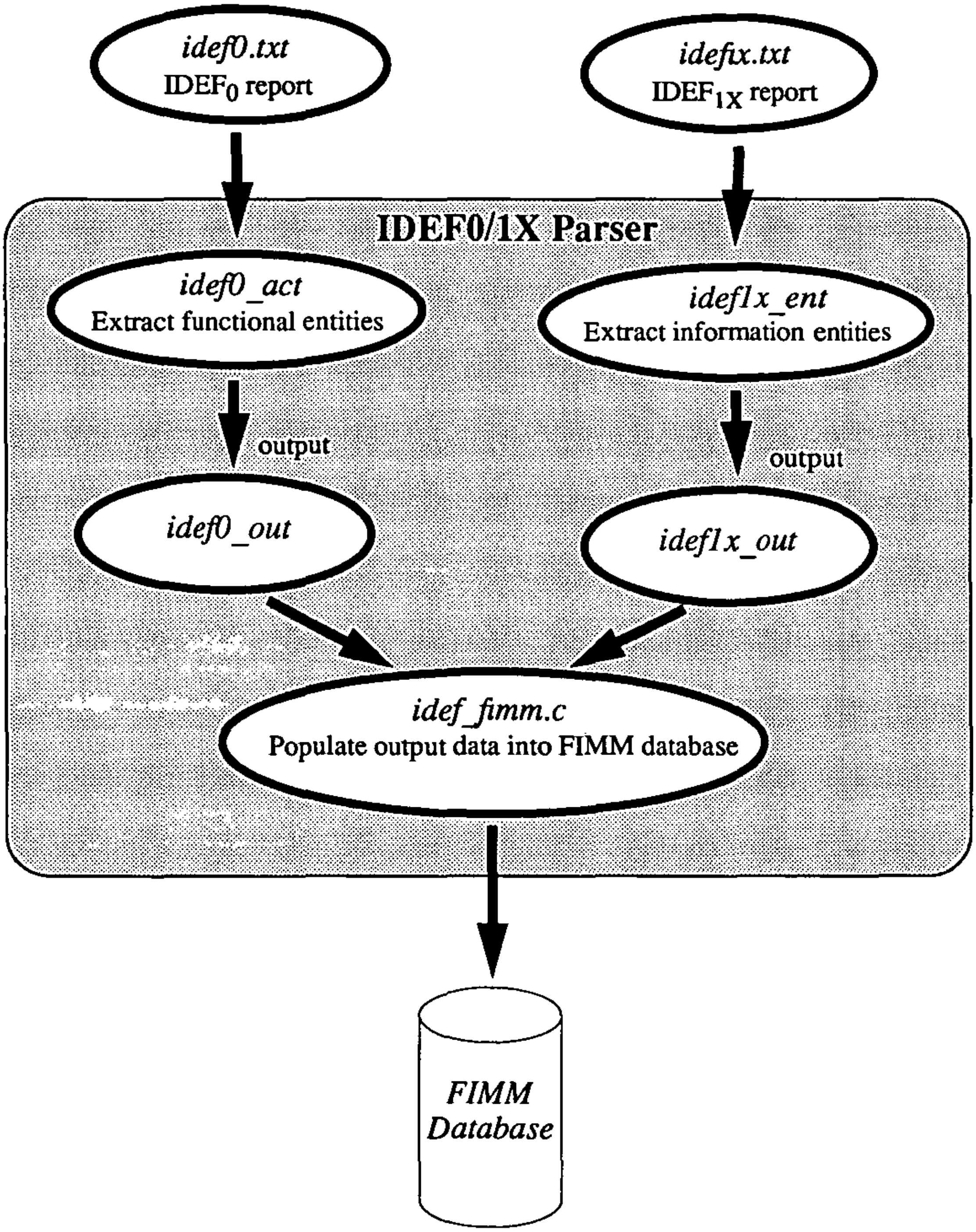
3. Program listing for idef1x_ent

4. Listings for idef0_out

5. Listings for idef1x_out

6. Program listing for idef_fimm.c

1. Program structure for $IDEF_{0/1X}$ Parser



2. Program listing for idef0_act

```
%%
int space_count = 1;
int field_count = 0;
int ch=0x22;
Activity:+[ ] {space_count = 1;
 ++field_count;
}
Arrow: field_count = 0;
Diagram: field_count = 0;
\n {++space_count ;
 if (space_count ==2)
 printf("%c\n%c",ch,ch);
 }
\tfield_count = 0;
[^]{if(field_count > 0 )
 printf("%s",yytext);
 }
```

3. Program listing for idef1x_ent

```
%%
int field_count = 1;
int space_count = 1;
int ch=0x22;
[,]+[ ] {++field_count;
 if(field_count==3)
 printf(“ “);
 }
ENTITY {field_count = 1;
 space_count = 1;
  }
\n { ++space_count ;
 if (space_count ==2)
 printf(“%c\n%c”,ch,ch);
 }
[^]{if(field_count > 1 && field_count < 4)
 printf(“%s”,yytext);
 }
```

4. Listing for idef0_out

```
“
“[A0] Part manufacture with MCS “
“[A1] Pre-Planning”
“[A2] Planning for Manufacture”
“[A3] Manufacturing Control”
“[A21] Production Planning”
“[A23] Process Planning”
“[A24] Product Design (CAD/CAM)”
“[A22] Finite Capacity Scheduler”
“
```

5. Listing for ide1x_out

```
“
“IE1 ORDER ENTRY”
“IE2 PART MASTER-BOM”
“IE4 PROCESS PLAN”
“DE21 BOM CHILD”
“DE41 MFG OPERATION ASSIGNMENT”
“DE43 RESOURCE ASSIGNMENT”
“DE42 MFG FACILITY ASSIGNMENT”
“IE5 MFG CELL CONFIGURATION”
“IE6 RESOURCE”
“IE3 ENGINEERING RESOURCE”
“IE7 CUSTOMER”
“IE8 SCHEDULE”
“IE9 SHOP FLOOR STATUS”
“IE10 MANUFACTURING FACILITY”
“IE11 SUPPLIER”
“DE101 PERSONNEL”
“DE102 MACHINE”
“
```

6. Program listing for idef_fimm.c

```
/* Populating extracted information from IDEF0 & IDEF1X into FIMM database*/

#include <stdio.h>
#include “local_incl.h”

FILE *temp_file;
char column[40];
char file_line[132];
char file_name[60];

extern char result_string[VERY_LONG_STRING];


main()
{
 int i,j,k;

char command[SMAL_STRING];
char val1[SMAL_STRING], val2[SMAL_STRING];
char help[MEDIUM_STRING], help2[MEDIUM_STRING];
FILE *drive;
char file_rest[FILE_LINE_LEN];

char file_path[MEDIUM_STRING];
char temp_sel[MEDIUM_STRING];
char temp_sel2[MEDIUM_STRING];
char idef0[MEDIUM_STRING];
char idef1X[MEDIUM_STRING];

strcpy(file_path,”/home2/sandra/valdew/oracle_c/fim/idef”);
sprintf(temp_sel,”%s/idef0_out_temp”,file_path);
sprintf(temp_sel2,”%s/idef1x_out_temp”,file_path);
sprintf(idef0,”%s/idef0_out”,file_path);
sprintf(idef1X,”%s/idef1x_out”,file_path);


 mw_connect(“mcc21”, “m”, “/home2/sandra/valdew/oracle_c/fim/idef/objects_idef.txt”);

printf(“\nEnter command (press RETURN for command list): “);
gets(command);
while((strcmp(command, “quit”))&&(command[0]!=’q’))
{
 if((command[0] == ‘f’)&&(command[1]==’0’))
  {
 /* Insert functions derived from IDEF0 into FIMM */
 system(“clear”);
printf(“\n\n\n\n”);
printf(“Populating FIMM Database (IDEF0 functions)\n\n”);
mw_query(“idef0_out_table”,””,temp_sel);
 if((drive = fopen(temp_sel,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel);
 file_rest[0] = ‘\0’;
```

```
while(file_reader(val2, drive, file_rest))
{
printf(“\n ----> %s”,val2);
sprintf(help,”’act’,’%s’,’new’”,val2);
mw_insert(“flat_idef0”,help);
sprintf(help,”’%s’,’s_p’,’ ‘”,val2);
mw_insert(“act_idef0_sp”,help);
sprintf(help2,”’%s’,’e_p’,’ ‘”,val2);
mw_insert(“act_idef0_ep”,help2);
mw_update(“idef0_fimm”,”answer=’*’”,””);
}
fclose(drive);
mw_commit();
printf(“\n\n\n”);
}
else if((command[0] == ‘f’)&&(command[1]==’1’))
 {
/* Insert entities derived from IDEF1X into FIMM */
system(“clear”);
printf(“\n\n\n\n”);
printf(“Populating FIMM Database (IDEF1X entities)\n\n”);
mw_query(“idef1X_out_table”,””,temp_sel);
if((drive = fopen(temp_sel2,”r”)) == NULL)
printf(“\nError, open %s.”,temp_sel2);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
{
printf(“\n ----> %s”,val1);
sprintf(help,”’dm’,’%s’,’new’”,val1);
mw_insert(“flat_idef1X”,help);
mw_update(“idef1X_fimm”,”answer=’*’”,””);
}
fclose(drive);
mw_commit();
printf(“\n\n\n”);
 }
else if((command[0] == ‘r’)&&(command[1]==’0’))
 {
system(“clear”);
printf(“\nPlease Wait......”);
printf(“\n\tIDEF0 ---> FIMM Parser is working\n\n”);
system(“runlex0”);
/* Insert functions derived from IDEF0 into IDEF0_OUT Table */
if((drive = fopen(idef0,”r”)) == NULL)
printf(“\nError, open %s.”,idef0);
file_rest[0] = ‘\0’;
while(file_reader(val1, drive, file_rest))
{
sprintf(help,”’%s’”,val1);
mw_insert(“idef0”,help);
}
fclose(drive);
mw_commit();
printf(“\n\n\n”);
}
else if((command[0] == ‘e’)&&(command[1]==’0’))
 {
system(“clear”);
system(“runform IDEF0_FIMM mcc21/m”);
mw_commit();
sprintf(help,“”);
mw_delete(“del_idef0_out”,help);
mw_commit();
 }
 else if((command[0] == ‘r’)&&(command[1]==’1’))
  {
 system(“clear”);
printf(“\nPlease Wait......”);
printf(“\n\tIDEF1X ---> FIMM Parser is working\n\n”);
system(“runlex1x”);
/* Insert functions derived from IDEF1X into IDEF1X_OUT Table */
 if((drive = fopen(idef1X,”r”)) == NULL)
 printf(“\nError, open %s.”,idef1X);
 file_rest[0] = ‘\0’;
 while(file_reader(val1, drive, file_rest))
 {
 sprintf(help,”’%s’”,val1);
 mw_insert(“idef1X”,help);
 }
 fclose(drive);
 mw_commit();
 printf(“\n\n\n”);
 }
 else if((command[0] == ‘e’)&&(command[1]==’1’))
  {
system(“clear”);
system(“runform IDEF1X_OUT mcc21/m”);
mw_commit();
sprintf(help,“”);
mw_delete(“del_idef1X_out”,help);
mw_commit();
 }
 else if((command[0] == ‘c’)&&(command[1]==’t’))
 {
mw_commit();
 }
 else if((command[0] == ‘r’)&&(command[1]==’b’))
 {
mw_rollback();
 }
 else
 {
 printf(“\14\nAvailable commands are:”);
 printf(“\n ‘r0’ IDEF0 --> FIMM Parser”);
 printf(“\n ‘e0’ IDEF0-FIMM Parser Editor”);
 printf(“\n ‘f0’ Populate IDEF0 function(s) into FIMM database”);
 printf(“\n\n ‘r1’ IDEF1X --> FIMM Parser”);
 printf(“\n ‘e1’ IDEF1X-FIMM Parser Editor”);
 printf(“\n ‘f1’ Populate IDEF1X entities into FIMM database”);
 printf(“\n\n ‘ct’ commit”);
 printf(“\n ‘rb’ rollback”);
 printf(“\n ‘q’ commit & quit”);
 printf(“\nCommand (%s) unknown. Try again.\n”,command);
 }
 printf(“\nEnter command (press RETURN for command list): “);
 gets(command);
```

```
}
printf("\n\n");
mw_commit();
mw_disconnect();
}
```

Meta-file definition

Name of meta-file : objects_idef.txt

Definition of database-objects: (for access_ora)

“flat_idef0” object name“flat_table” table name
“type, name, status”fields““ where condition

“idef0” object name“idef0_out” table name
“activity”fields““ where condition

“del_idef0_out” object name“idef0_out” table name
“activity, answer, act_id”fields“answer=’N’” where condition

“idef0_out_table” object name“idef0_out” table name
“act_id”fields“answer=’Y’” where condition

“idef0_fimm” object name“idef0_out” table name
“answer, act_id”fields“answer=’Y’” where condition

“flat_idef0” object name“flat_table” table name
“type, name, status”fields““ where condition

“idef1X” object name“idef1X_out” table name
“activity”fields““ where condition

“idef1X_out_table” object name“idef1X_out” table name
“activity”fields“answer=’Y’” where condition

“del_idef1X_out” object name“idef1X_out” table name
“activity, answer, act_id”fields“answer=’N’” where condition

“idef1X_fimm” object name“idef1X_out” table name
“answer”fields“answer=’Y’” where condition

“flat_idef1X” object name“flat_table” table name
“type, name, status”fields““ where condition

“act_idef0_sp” object name“activity_info_table” table name
“activity, type, information”fields““ where condition

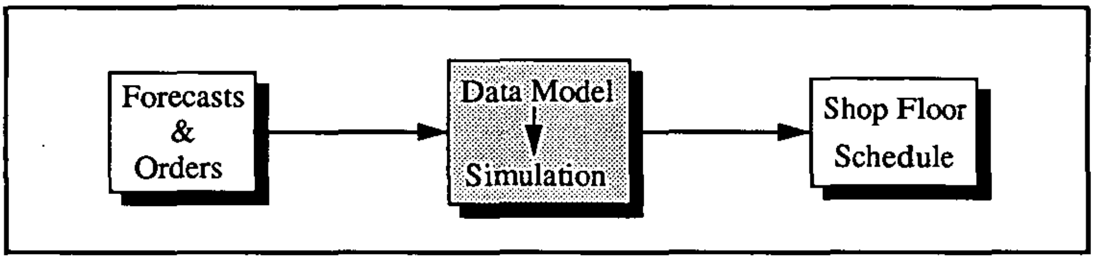
“act_idef0_ep” object name“activity_info_table” table name
“activity, type, information”fields““ where condition

APPENDIX XIX

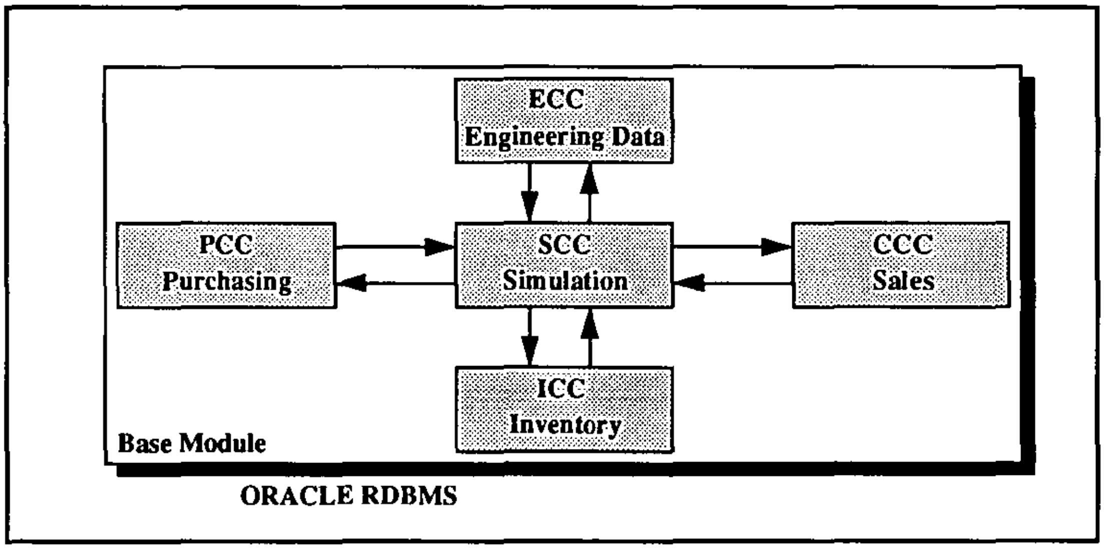
Overview
MCC CAPM software package

MCC

MCC (Manufacturing Control Code) is a commercially available computer aided production management application software supplied by John Brown Systems PLC. It is for the control of production, especially of material movement. The control is achieved by utilising a realistic model of the production process. The model focuses on the shop floor, and covers the resources available (labour and machines), and the production process (what operations must be performed, their order, how they are linked together, their standard times etc.). This model can be loaded with a combination of orders and forecasts of expected future demands. Running the production simulator produces a schedule taking into account all the production constraints, material requirements remain associated with resource capacity. On this basis MCC produces at a single pass, realistic production and material requirement plans, as illustrated below.



MCC comprises the following six major modules which use data residing in a common ORACLE RDBMS :



The central module SCC is the production simulation module, and provides a tool for finite capacity planning and shop floor scheduling. The module details facility and resource availability, shift patterns, and the job load.

Details of bills of material, product routings, and production resources are maintained in the engineering control module (ECC).

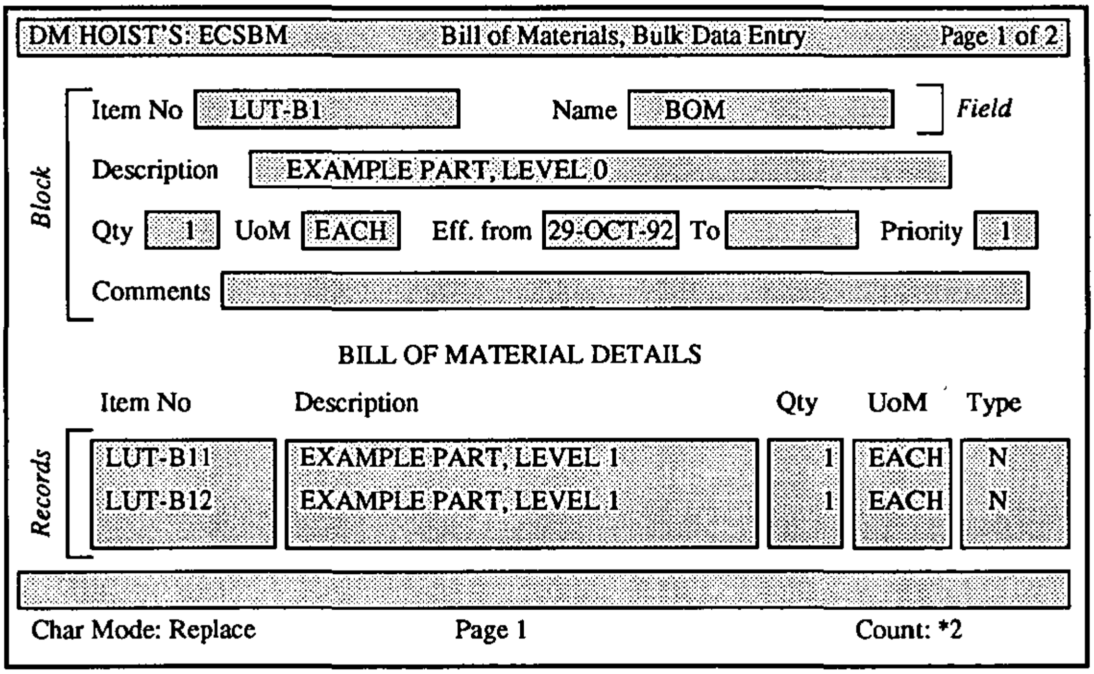
The purchasing functions covering stock, non-stock, and service orders are controlled by the purchasing control module (PCC).

Sales and order processing functions are to be carried out using the customer control code module (CCC) whilst the inventory control code module (ICC) maintains stock and material requirement data across multiple stores and their multiple locations.

The system is tied together by software which controls the database tables, report functions, menu systems, and user authority codes.

User Interface

The proprietary user interface of MCC is written in ORACLE RDBMS based SQL*Forms, which is an application development environment package. A form is a fill-in-the-blanks template displayed on a computer screen that allows the user to enter, update and query information in a database. As illustrated in the example below, forms are composed of blocks, records and fields.



DM HOIST'S: ECSBM Bill of Materials, Bulk Data Entry Page 1 of 2

Item No LUT-B1 Name BOM

Description EXAMPLE PART, LEVEL 0

Qty 1 UoM EACH Eff. from 29-OCT-92 To   Priority 1

Comments

BILL OF MATERIAL DETAILS

Item No	Description	Qty	UoM	Type
LUT-B11	EXAMPLE PART, LEVEL 1	1	EACH	N
LUT-B12	EXAMPLE PART, LEVEL 1	1	EACH	N

Char Mode: Replace Page 1 Count: *2

MCC form for its Bill of Materials option - ECSBM

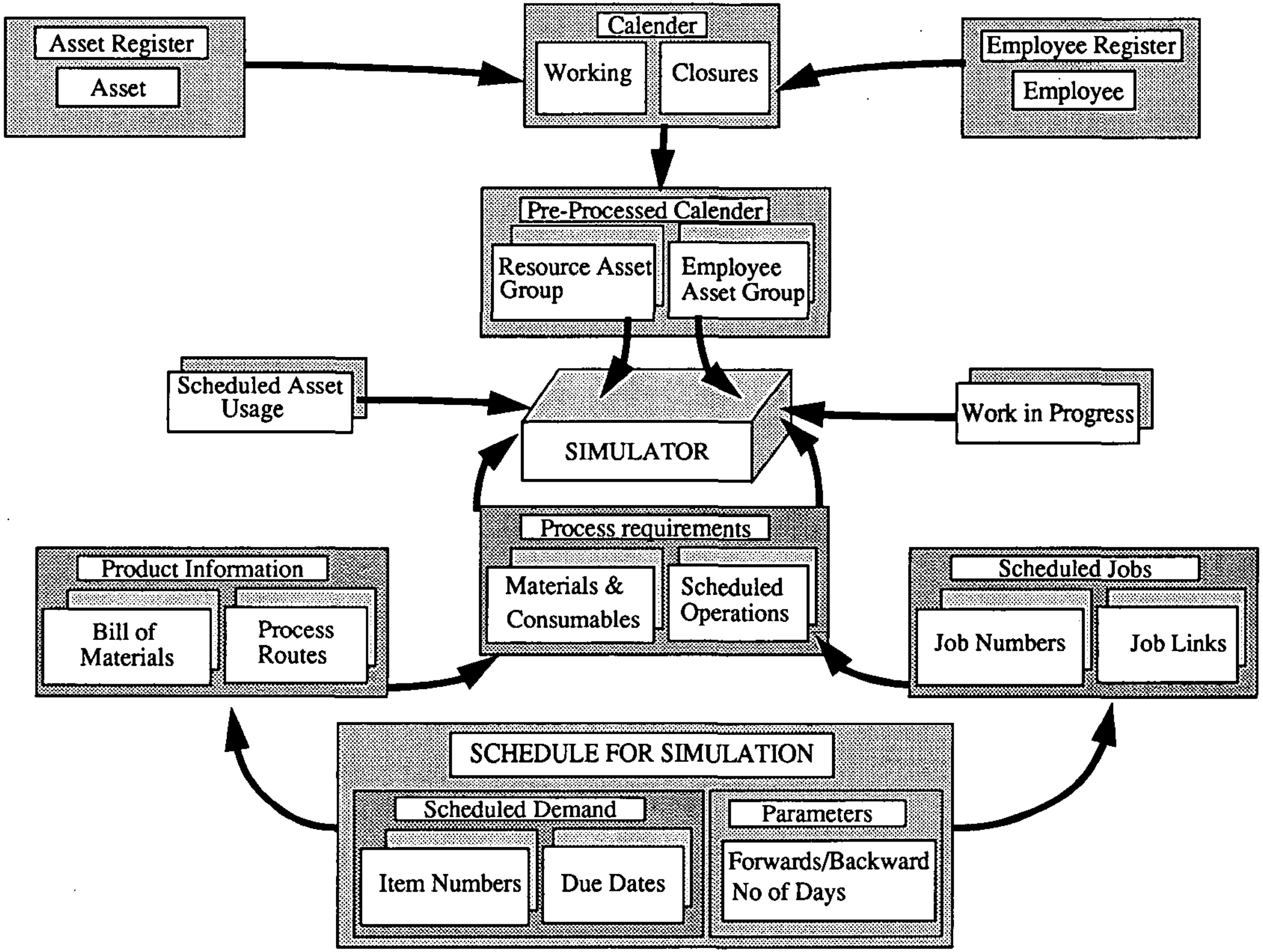
Block: A group of related fields on a form.

Record: The data from a row in a database or non-database table.

Field: An area on the screen that can display a value or accept an input value. A field normally represents a column form a database table.

In general, MCC is written in SQL*Forms and it consists of several forms and program routines.

INFORMATION REQUIREMENTS OF MCC



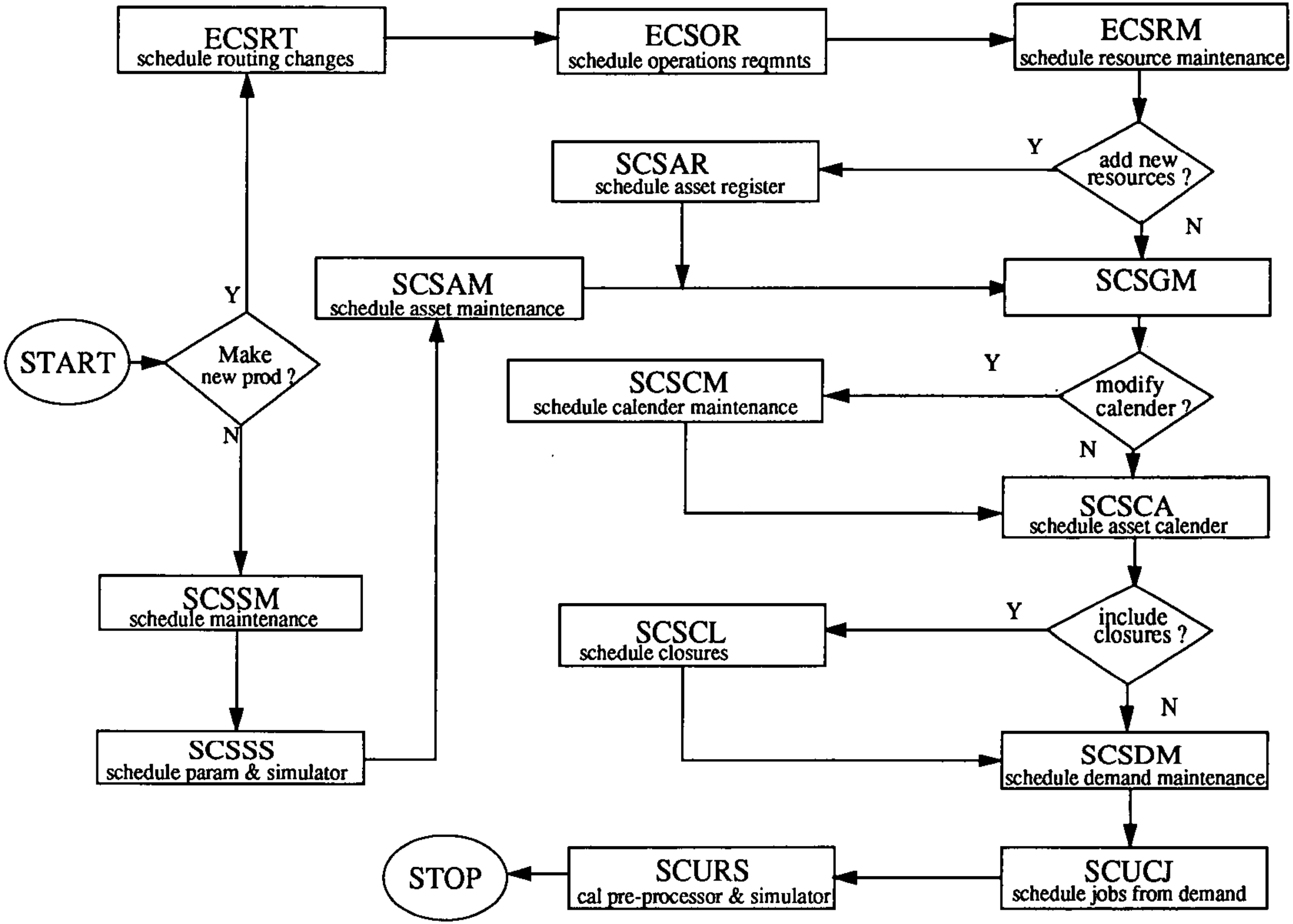
Operational Steps in MCC

Module	Description	Remarks
MCSIM	Items registration : - Components/raw materials/tools sub-components	
ECSBM	Bill of materials (2 levels build up)	
* if any changes use ECSEC: password - Damian Type - N Origin - Anything	Changes & updates to BOM	To get indented BOM report call ECRIB
* for queries use ECSIB which shows indeneted BOM	Query on BOM	
ECSRT	Entry & update on routing	Need to build "bottom-up" Run **proc_routes** for routing report
* Call ECSOR within ECSRT (Esc 8) for operations requirement maintainance.	Page 2 for BOM assignment	
ECSRM	Resource maintenance	
SCSSM	Create Schedule & Parameter setting for schedule run	Note the series must be assigned.
SCSAR	Asset register	Real machines with inventory no.
SCSAM	Schedule Asset maintenance - Resource to Asset allocation for schedule.	Resources need general grouping.
SCSGM	Asset grouping	
SCSCA	Calendar allocation to schedule	
SCSCM	Calendar definition	
SCSCL	Calendar closure maintenance	Block off period e.g. machine maintenance & labor availability. Need to run this module if changes are done to the calendar.
➡ SCSDM	Demand maintenace	Akin to schedule log.
SCSCJ	Create job from demand.	Assign routing to job.
SCUCJ	Create job links.	Check for corresponding links. If don't exist delete SQl> **Update sch_job set stat = NULL where sch_ident = 1361 and sch_job_no = 6758 and sch_job_ser_name = 'Swivel'; Commit;**
SCSJM	Schedule job maintenace	Also to query on assigned routing. Can block off unwanted jobs by changing status to 'Y' (non-Schedulable)
SCURS	Run calendar preprocessor.	
SCCSM	Schedule run.	If schedule need to be adjusted go to SCSSM to change SCH_PERIOD in parameter. Further modifications can be done in ECSRT by changing contigous (MC) to non-contigous (MN) e.g. those > 10 hrs of operation in the schedule.
➡ SCMSR	Simulation reports : SCRJL & SCRWL	
SCSWP	Record work in progress.	Page 2 for further details.

To create new demands

PCSIS	➡	**Item sourcing**
MCSOE		**External Organization information**
CHSOR		**Sales Order**

STEPS TOWARDS RUNNING MCC



APPENDIX XX

Program listings of ‘alien application shell’ for MCC

‘Alien application shell’ for MCC

```
/*Program :new_mcc_mw_1.c*/
/*Created:11 - 06 - 93*/

/*MCC & CIM-BIOSYS application Main Interface.*/

/* Functions:
mw_die() Terminates the CBS-Application, MCC and the process connected to Oracle.

mw_quit() Calls mw_die() if no MCC-Option is active.

mw_tick_tock() Displays visible clock. Also used to
Start MCC and a process connected to Oracle.
Displays the cursor in the MCC screen.
Count the slow_down_count variable.
Initialise the message buffer.

app_user_disp_data()
app_user_disp_req()

copy_mess(to_mes, from_mess)
Copies a message of the type mess_data.

init_mes_buf() Initialises the message buffer.

put_mes(mes) Copies the message mes in the message buffer. If the message buffer is full, the message is not
copied and the result of the function is 0. Otherwise the result is 1.

get_mes(message) Gets the next message from the message buffer. If the message buffer was empty, the result of
the function is 0. Otherwise the result is 1.

next_message() Starts the processing of the next message.

receiv_message() Decides if the received message can be processed or if it has to be stored in the message buffer.
*/

#define PERMIT0x7fff

#include “local_incl.h”
#include<string.h>
#include“global_inc.def”/* Defs for all modules. */
#include“wind_inc.def”/* Defs for window interface modules*/
#include“new_mcc_wind_inc.def”/* Defs for window modules.*/
#include“app_user_inc.def”/* Defs for user application modules.*/
#include“app_serv_inc.def”/* Defs for app_serv task modules. */
#include“use_mcc_special.def”/* Defs for use_mcc */
/* and run_mcc only */
#include“demo_comm.def”/* Defs for communication. */

#include <stdio.h>
#include <ctype.h>
#include <string.h>

#ifndef NULL
```

```
#define NULL 0
#endif

char state_start = 0;
extern char cursor_state;
extern char slow_down_count;

extern int main_state;
extern int mcc_p1[2];

struct mess_data message_buffer[MES_BUF_LEN];
int mcc_busy_state_1 = 0;
int point_to_write, point_to_read;
struct mess_data cbs_message;
struct mess_data act_message;
int mess_data_len;

void init_mes_buf(), copy_mess();
char put_mes(), get_mes();
void next_message(), receiv_message();

/*Functions external to this module.*/

/*Variables external to this module.*/
externstructs_frame_desc*wind_data[NO_WINDOWS] ;
structs_app_link
 {charname[21] ;
 }
extern app_links[MAX_APP_LINKS] ;

/*Functions internal to this module*/
intmw_die() ;


long intnow ;
struct s_cbs_req_datacbs_req_data ;



/*voidmw_quit()*/
/*-----------------*/
/*Panel button routine to quit the application. */
/*return value*/
/*void.*/

voidmw_quit()
{register intresult ;
charbuffer[81] ;
 if(no_option_activ())
 {
pi_reg_func(mw_die,5) ;
sprintf(buffer,"%s TERMINATING In 5 SECS.",g_proc_title) ;
window_set(wind_data[M_WINDOW]->frame,FRAME_LABEL,buffer,0) ;
 }
 else
wi_put_str_mess(M_WINDOW,MW_DEF_LAYOUT,MW_ANW,
 "Exit current screen before leaving MCC.");
}

/*intmw_die()*/
/*-----------------*/
/*Terminates the current process.*/
/*return value*/
/*FALSEto stop the pulling of this function*/

intmw_die(t_count)
intt_count ;
{register intresult ;

quit_mcc();
quit_remote_new_mcc("1");

result = it_term_this_proc() ;
return(FALSE) ;
}



/*intmw_tick_tock(t_count)*/
/*-------------------------*/
/*Display a visible clock tick in the main interface window and times*/
/*out the window if necessary*/
/*intt_countcurrent tick count*/
/*return value*/
/*TRUEso that polling of this function continues */

intmw_tick_tock(t_count)
int t_count;
 {register inttmp,idx ;
 staticstat_count = 1 ;

if(!state_start)
{
 state_start++;
 start_mcc();
 start_new_mcc_ora();
 init_mes_buf();
 mess_data_len = sizeof(struct mess_data);
}

if(slow_down_count < 5) slow_down_count++;

if(cursor_state) cursor_state = 0; else cursor_state = 1;
mcc_cursor();

/*put a tick/tock message into appropriate field.*/

wi_tick_display(M_WINDOW,MW_DEF_LAYOUT,MW_TICK,t_count) ;

return(TRUE) ;
}

/*intapp_user_disp_data(cmd_flag,user_cmd_data_ptr)*/
/*-----------------------------------------------------*/
/*Display the contents pointed by user_cmd_data_ptr either as a received*/
```

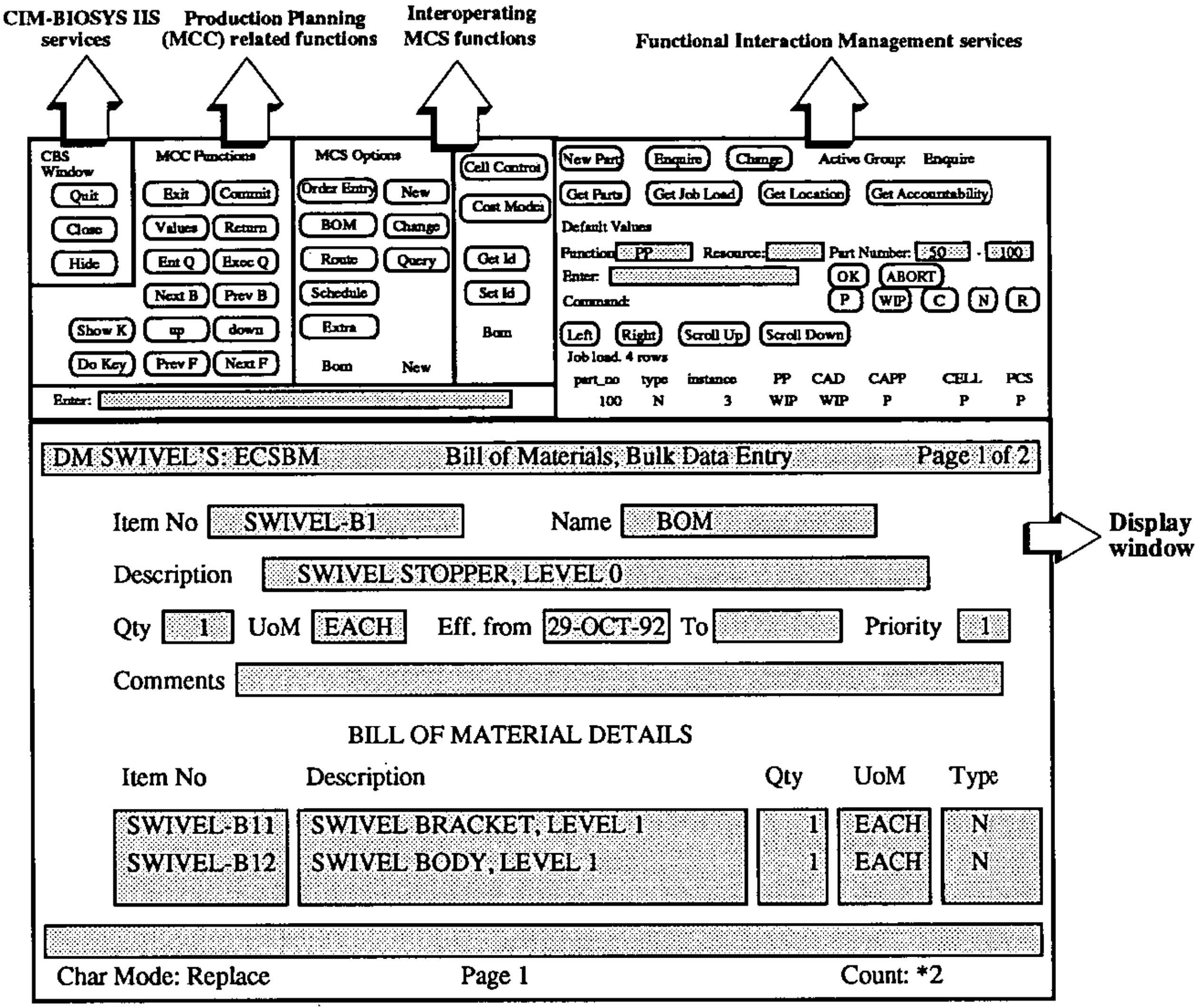
```
/*command or response according to cmd_flag.*/
/*intcmd_flag*/
/*command flag (TRUE for command, FALSE for response)*/
/*struct s_cbs_cmd_data*user_cmd_data_ptr ;*/
/*pointer to user command data*/
/*return value*/
/*OK.*/
/*When cmd_flag = TRUE, the start_copmmand function is called.*/

intapp_user_disp_data(cmd_flag,user_cmd_data_ptr)
intcmd_flag ;
struct s_cbs_cmd_data*user_cmd_data_ptr ;
{register inttmp,idx,result,len,cnt ;
charbuffer[81],*name_ptr ;
long intnow ;

result = OP_FAILED ;
now = time(NULL) ;

printf(“\napp_user_disp_data”);

if ( cmd_flag == 0 )
  {
  name_ptr = user_cmd_data_ptr->sarg1 ;
  }
 else
  {
  name_ptr = user_cmd_data_ptr->sarg2 ;
  }

if (cmd_flag)
{
 copy_mess(&cbs_message, user_cmd_data_ptr->data_ptr);
 receiv_message();
}

if ( user_cmd_data_ptr->cmd_code== CBS_EST_LINK &&
user_cmd_data_ptr->status == 0 )
 {
tmp = si_find_link_entry(name_ptr) ;
if ( tmp == NOT_FOUND )
 {tmp = si_find_link_entry(“”) ;
if ( tmp != NOT_FOUND )
 {sprintf(app_links[tmp].name,”%.20s”,name_ptr) ;
  }
  }
 }
if ( user_cmd_data_ptr->cmd_code == CBS_TERM_APP )
 {tmp = si_find_link_entry(name_ptr) ;
if ( tmp != NOT_FOUND )
 {
sprintf(app_links[tmp].name,”%.20s”,””) ;
  }
 }
if ( user_cmd_data_ptr->cmd_code == CBS_SEND_APP &&
cmd_flag == FALSE &&
user_cmd_data_ptr->status != 0 )
 {tmp = si_find_link_entry(name_ptr) ;
if ( tmp != NOT_FOUND )
 {
sprintf(app_links[tmp].name,"%.20s","") ;
  }
 }
sprintf(buffer,"Links : None.") ;
len = 8 ;
for ( idx = 0,cnt = 0 ; (idx < MAX_APP_LINKS) && (cnt < 4) ; idx++ )
 {if ( app_links[idx].name[0] != 0 )
  {sprintf(buffer + len,"%-10.10s",app_links[idx].name) ;
  cnt++ ;
  len += 10 ;
  }
 }
wi_put_str_mess(M_WINDOW,MW_DEF_LAYOUT,MW_M_LINKS,buffer) ;
if ( cnt == 0 && strcmp(g_proc_title,"tom") != 0 )
 {
pi_reg_func(mw_die,5) ;
sprintf(buffer,"%s TERMINATING In 5 SECS.",g_proc_title) ;
window_set(wind_data[M_WINDOW]->frame,FRAME_LABEL,buffer,0) ;
 }

result = OK ;
return(result) ;
}

/*intapp_user_disp_req(cbs_req_data_ptr)*/
/*-----------------------------------------*/
/*Display the contents of a CMBSYS request from us to ASP.*/
/*struct s_cbs_req_data*cbs_req_data_ptr ;*/
/*pointer to CMBSYS request*/
/*return value*/
/*OK.*/

intapp_user_disp_req(cbs_req_data_ptr)
struct s_cbs_req_data*cbs_req_data_ptr;
{register intidx,tmp,result ;
charbuffer[201] ;
long intnow ;

now = time(NULL) ;

cbs_req_data_ptr->data_ptr[cbs_req_data_ptr->data_len] = 0 ;
return(OK) ;
}

void copy_mess(to_mes, from_mes)
char *to_mes;
char *from_mes;
{
 int i;

 for(i = 0; i<mess_data_len; i++)
 to_mes[i] = from_mes[i];
 }

void init_mes_buf()
 {
```

```
 int i;
 for(i=0;i<MES_BUF_LEN;i++)
 {
message_buffer[i].sender[0] = ‘\0’;
 }
 point_to_write = 0;
 point_to_read = 0;
}

char put_mes(mes)
struct mess_data *mes;
{
 if(!message_buffer[point_to_write].sender[0])
 {
copy_mess(&message_buffer[point_to_write], mes);
point_to_write = (++point_to_write)%MES_BUF_LEN;
return(1);
 }
 else return(0);
}

char get_mes(message)
struct mess_data *message;
{
 if(message_buffer[point_to_read].sender[0])
 {
copy_mess(message, &message_buffer[point_to_read]);
message_buffer[point_to_read].sender[0] = ‘\0’;
point_to_read = (++point_to_read)%MES_BUF_LEN;
return(1);
 }
 else return(0);
}

void next_message()
{
 struct mess_data help;

 if(get_mes(&help))
 {
process_message(&help);
 }
 else
 {
act_message.sender[0] = ‘\0’;
mcc_busy_state_1 = 0;
 }
}

void receiv_message()
{
 printf(“\nMessage received.”);
 help_print(&cbs_message);

printf(“\nBusy: %d.”,mcc_busy_state_1);

 if(cbs_message.sender[0])
 {
```

```
 if(mcc_busy_state_1)
 put_mes(&cbs_message);
 else
 {
process_message(&cbs_message);
 }
 }
}
```

APPENDIX XXI

Services and Options offered via User Interface



Bom New

Bom

Enter:

Active Group: Enquire

Default Values

Function: PP Resource: Part Number: 50 - 100

Enter:

Command:

Job load. 4 rows

part_no	type	instance	PP	CAD	CAPP	CELL	PCS
100	N	3	WIP	WIP	P	P	P

DM SWIVEL'S: ECSBM Bill of Materials, Bulk Data Entry Page 1 of 2

Item No: SWIVEL-B1 Name: BOM

Description: SWIVEL STOPPER, LEVEL 0

Qty: 1 UoM: EACH Eff. from: 29-OCT-92 To: Priority: 1

Comments:

BILL OF MATERIAL DETAILS

Item No	Description	Qty	UoM	Type
SWIVEL-B11	SWIVEL BRACKET, LEVEL 1	1	EACH	N
SWIVEL-B12	SWIVEL BODY, LEVEL 1	1	EACH	N

Char Mode: Replace Page 1 Count: *2

The user interface consists of an input/ output display window, an entry field and several specially configured buttons. The window displays the output of MCC to the user. It has the same format as the original output of MCC. Only a flashing '*' was added to mark the current cursor position.

The functions of the buttons can be divided in three groups:

- MCC functions to edit a form or commit changes.
- Call a MCC option.
- Cross reference MCC data with FIMM data.

If the user wants to enter data in the MCC forms, the entry field is used instead of entering the data directly in the form. This data input will be sent to MCC when one of the MCC function buttons is pressed.

The buttons for the MCC function offer only those functions which are used frequently when working with MCC. They include buttons for moving around in a form, committing changes and leaving a form. As there are 44 functions offered by MCC, they can't all be offered via buttons. To call a function which is not directly offered by a button, the user can enter the name of the desired function in the entry field and press the button 'Do Function'. The button 'Show Functions' will display a list of all available functions. Note that the list of available functions depends on the actual form and field.

The buttons to call MCC options make easier access to some of the MCC options. Instead of going through the menu structure of MCC, those options can be selected directly by pressing the appropriate button. The offered options are sufficient to build a production simulation module, enter product information and orders and to produce schedules by running the production simulator.

The options are divided into five groups:

- Bill of material
  The options of this group allow to define a bill of material that contains all the stock items required to produce a manufactured item.
- Routings
  These options are used to create and maintain information about the method of manufacture for an item.
  i. The required operations to produce the item.
  ii. The order these operations are executed in order to ensure that the item is made in the correct sequence.
  iii. The requirements of these operations, in terms of resource and stock items.
- Schedule Model
  The shop floor model for the simulation is build with these options. They allow to specify the availability of machines and labour.
- Schedule Run
  The workload, described as a number of jobs is defined with the options in this group. In addition, they allow to run the simulator and to feed back data about completed operations and jobs.
- Extra
  The options in this group are very specific to MCC, but they are needed to build and run a production simulation. They allow to enter general information like available assets of employees without relating them to the production simulation model.

The last group of buttons are provided to relate data FIMM to data in MCC. Each part can be related to bill of material, routing and schedule job information. To assign information to parts, the status of that part must be displayed in the FIMM output field and the MCC data must be displayed in the MCC screen. The button 'Set Id' relates the information together.

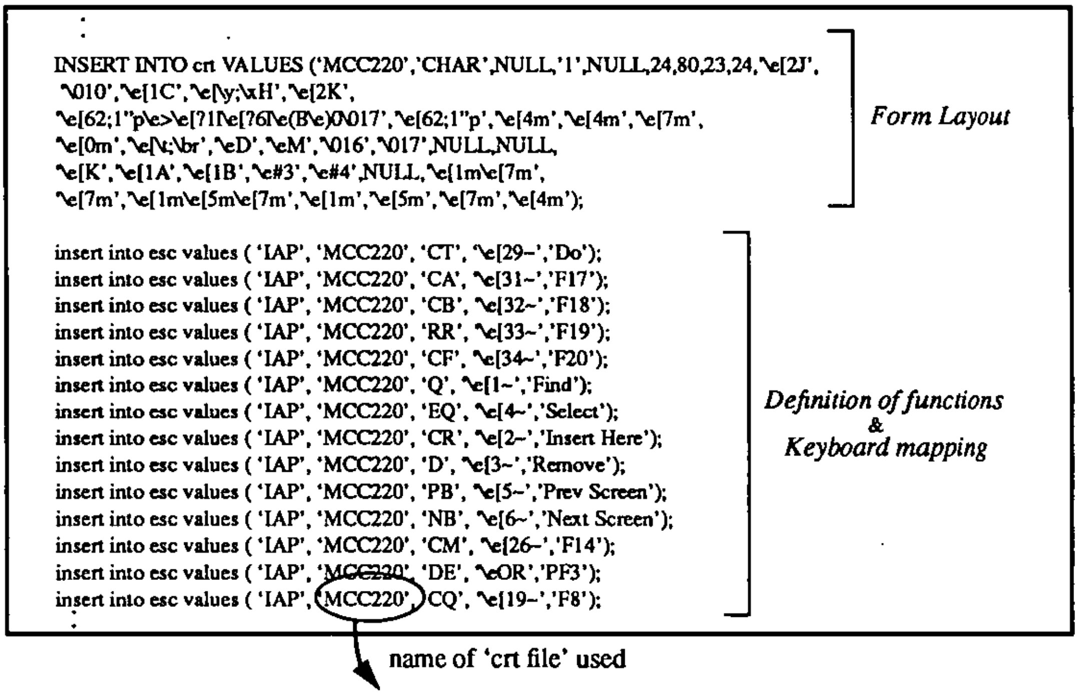
Buttons have also been specially configured to activate and interact with the Cell Controller and the Cost Modeller through the user interface.

APPENDIX XXII

Communication between MCC and the User Interface

MCC expects input consisting of readable character strings and esc sequences, as illustrated below. The character strings are used to enter data in the fields of a form. The esc sequences invoke special functions e.g. to edit a form or to retrieve values or to commit changes. The output of MCC consists of character strings and esc sequences too. The character strings contain the text, displayed on the screen and the esc sequences specify the format and the location of that text.

The information about the meaning of those esc sequences is stored in a special file, called 'crt file' (please refer below for illustration) [ORACLE RDBMS 1991]. By using different crt files, it is possible to run SQL*Forms applications like MCC on different terminals. To make the interpretation of the MCC output easier, a special crt file was created. It contains esc sequences with format specifications which can be represented in the input/output display window of the user interface over CIM-BIOSYS IIS. It is also necessary to know the structure of those options and the related forms. For example the number of blocks in a form, the sequence of fields and the error messages which can occur, when entering data in a field.



```
:
INSERT INTO crt VALUES ('MCC220','CHAR',NULL,'1',NULL,24,80,23,24,'\e[2J',
 '\010','\e[1C','\e[\y;\xH','\e[2K',
'\e[62;1"p\e>\e[?1l\e[?6l\e(B\e)0\017','\e[62;1"p','\e[4m','\e[4m','\e[7m',
'\e[0m','\e[\t;\br','\eD','\eM','\016','\017',NULL,NULL,
'\e[K','\e[1A','\e[1B','\e#3','\e#4',NULL,'\e[1m\e[7m',
'\e[7m','\e[1m\e[5m\e[7m','\e[1m','\e[5m','\e[7m','\e[4m');

insert into esc values ( 'IAP', 'MCC220', 'CT', '\e[29~','Do');
insert into esc values ( 'IAP', 'MCC220', 'CA', '\e[31~','F17');
insert into esc values ( 'IAP', 'MCC220', 'CB', '\e[32~','F18');
insert into esc values ( 'IAP', 'MCC220', 'RR', '\e[33~','F19');
insert into esc values ( 'IAP', 'MCC220', 'CF', '\e[34~','F20');
insert into esc values ( 'IAP', 'MCC220', 'Q', '\e[1~','Find');
insert into esc values ( 'IAP', 'MCC220', 'EQ', '\e[4~','Select');
insert into esc values ( 'IAP', 'MCC220', 'CR', '\e[2~','Insert Here');
insert into esc values ( 'IAP', 'MCC220', 'D', '\e[3~','Remove');
insert into esc values ( 'IAP', 'MCC220', 'PB', '\e[5~','Prev Screen');
insert into esc values ( 'IAP', 'MCC220', 'NB', '\e[6~','Next Screen');
insert into esc values ( 'IAP', 'MCC220', 'CM', '\e[26~','F14');
insert into esc values ( 'IAP', 'MCC220', 'DE', '\eOR','PF3');
insert into esc values ( 'IAP', 'MCC220', 'CQ', '\e[19~','F8');
:
```

Function Code	Function Description	Escape Sequence	Comments
CT	Commit Transaction	\e[29~	Do
CA	Clear Form / Rollback	\e[31~	F17
CB	Clear Block	\e[32~	F18
RR	Clear Record	\e[33~	F19
CF	Clear Field	\e[34~	F20
Q	Execute Query	\e[1~	Find
EQ	Enter Query	\e[4~	Select
CR	Create Record	\e[2~	Insert Here
D	Delete Record	\e[3~	Remove
PB	Previous Block	\e[5~	Previous Screen
NB	Next Block	\e[6~	Next Screen
CM	Insert / Replace	\e[26~	F14
DE	Display Error	\eOR	PF3
CQ	Count Query Hits	\e[19~	F8
:	:	:	:

With knowledge of the crt file, MCC functions can be offered through the user interface over CIM-BIOSYS IIS. The user can enter text in an entry field which is send to MCC. Specially configured buttons are provided to invoke the special functions for editing a form and committing changes. The output of MCC is scanned for the esc sequences that specify the format and location of the text strings. This information is used to display the MCC forms through the user interface over CIM-BIOSYS IIS via the input/output display window provided. Through this interface, the complete functionality of MCC is made available to users.