Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

# Design and Development of a Hybrid Control System for Flexible Manufacturing

A Thesis presented in partial fulfilment of the requirements
for the degree
of Master of Technology in
Manufacturing and Industrial Technology at
Massey University

Michael David Butler
1994

# ABSTRACT

Flexible Manufacturing Systems (FMS) appeared upon the manufacturing scene in the early 1970s, installations presently number in the thousands. However, many current installations in fact lack flexibility, do not operate in real-time and are prohibitively expensive. Therefore there are obvious benefits to be gained from making improvements to existing flexible manufacturing systems.

Research conducted for this thesis focused on two major areas. The implementation of the FMS control system on a SCADA package and the development of an auction based scheduling system. This entailed the development of a hybrid control model composed of three distinct layers; factory, cell and intelligent entity. Key portions of both the factory and cell controllers were then implemented so as to create a minimal system. This has been completed to the point where the auction algorithm has been implemented and tested in an appropriate framework.

In achieving the goals mentioned above a number of novel design concepts have been utilised. There are two which are most important, these are the use of low cost modules for the construction of a flexible co-operative manufacturing system, and the ability of this system to operate in a physically distributed area via a Local Area Network. Meaning it is inherently adaptable and resistant to failure. These novel design concepts were ingrained throughout the entire three layered control model.

It is felt that this research has succeeded in demonstrating the possibility of implementing a FMS control system on a low cost SCADA package using low cost software and computing elements. The ability of the distributed, auction-based approach to operate successfully within this system, has also been demonstrated through simulation.

# ACKNOWLEDGMENTS

I would like to acknowledge a number of members of the staff within the Production Technology Department whose assistance has proved invaluable; Prof Don Barnes, Len Chisholm, Dr Simon Hurley and Nola Babbage. I would also like to thank my fellow postgraduate students, especially Heather, Ian, Nigel, Mike and Kev all who have given me friendship and encouragement.

My supervisor Dr Saeid Nahavandi has over the last two years provided me with help and direction. Without his drive and dedication this research would never have been possible.

I would finally like to thank my friends and family both past and present who have supported, encouraged and provided me the wisdom to continue.

# TABLE OF CONTENTS

Chapter 5



Chapter 6

# LIST OF FIGURES AND TABLES

## List of Figures

## List of Tables

# CHAPTER 1

## INTRODUCTION

Table of Contents

## 1.1. Topic of Research

The research undertaken by the author is concerned with the design and development of a low cost, hybrid control system for flexible manufacturing, utilising SCADA software. A flexible manufacturing system (FMS), can be broadly defined (Nagarur 1992, Weston et al 1991) as a manufacturing system that is able to take advantage of changes in market requirements, manufacturing resources and technology. One approach to the development of FMSs is the use of a hybrid control architecture. The hybrid architecture is a combination of both heterarchical and hierarchical control architectures.

Through combining the vertical control structure of hierarchical control with the distributed decision making ability of heterarchical control, certain benefits can be achieved. This architecture is intended to negate the disadvantages of both control methodologies, whilst retaining the positive aspects. Supervisory control and data acquisition software or SCADA software is one possible method of implementing such a control methodology at a low cost, in comparison to higher cost existing manufacturing control system. This research, conducted at the Department of Production Technology at Massey University, is concerned with not only the design of an appropriate control model, but also with the implementation of the model. This research has focused primarily upon development of the software side, of the flexible manufacturing control system.

## 1.2. Scope of Research

The research conducted here concerns the development of a manufacturing system where flexibility has been maximised. The flexibility or adaptability of a manufacturing system allows it to adapt to change. The type of change affecting a manufacturing system can be said to consist of two components (Mandelbaum 1978, 1989), predictable or planned change and unpredictable or random change. A manufacturing system must be able to adapt to planned system changes but also should be able to cope with randomly distributed disturbances. A manufacturing system's ability to adapt to factory re-configuration and a high product mix as well as machine break down and late job alteration, will be directly reflected in an organisation's ability to satisfy its customers at a minimum cost. It is felt that this is especially true in the case of the New Zealand manufacturing industry. Not only does New Zealand have a small domestic market but increasingly the ability of New Zealand manufacturers to effectively compete in overseas markets is through targeting small niche markets and providing high quality at low cost.

To achieve a low cost implementation of a flexible manufacturing control system, two novel concepts were proposed. These were; the use of a hybrid distributed control methodology and low cost implementation resources such as SCADA software. Through using these two concepts it was hoped that a highly flexible manufacturing control system could be achieved relatively quickly and cheaply using SCADA software as an implementation backbone. One of the most significant components of the control architecture's implementation was the use of an auction-based scheduling system. The auction or bidding system utilised an opportunistic approach to the distribution of jobs amongst the various processing elements within the manufacturing system and has the potential to operate in real time. These two original concepts formed the focus of research in relation to this thesis, that is the design and implementation of a hybrid flexible manufacturing control system using SCADA software and the development of an auction based system for the scheduling of jobs within the factory.

Research in both areas followed a planned development cycle; design, implementation and testing. The aim of the design phase consisted of firstly investigating the topic, then developing a capable control model. The implementation phase was concerned with the implementation of the control model using the SCADA software. It was realised early on that the implementation of the entire control model was constrained by time. Therefore the focus of the implementation phase was to implement the control model in software to the point where the auction system could be tested in a realistic environment. The testing or experimental phase of this research focused upon evaluating whether or not it would be feasible to implement a hybrid flexible manufacturing system utilising SCADA software and if the auction based scheduling system was an appropriate method of real time scheduling in such an environment.

## 1.3. The Importance of Flexible Manufacturing

As many New Zealand companies know only too well, competition between organisations within similar domains both locally and internationally for the customer's dollar is becoming increasingly fierce. Major economic changes which began in New Zealand in 1984 with the removal of tariff protection, meant many multi national companies left New Zealand. Since then consecutive New Zealand governments have continued to pursue free market style economic reforms. This has resulted in a comparatively competitive domestic market and an export focused economy. In this new environment a number of organisational characteristics emerged as being critical to maintaining competitiveness. Flexible, efficient, customer driven organisations that were able to respond to market requirements quickly, were also the organisations that tended to perform the best in competitive situations.

The outcome of the GATT agreement in 1993 on international trade and tariffs is proof that pressure around the world is mounting for countries to reduce local market protection. Whilst a truly free global economy is still a long way off, it is felt that the general consensus of the UN and world leaders is that the gradual removal of trade barriers between countries, is beneficial in most cases to both local and global economies. Competition on a relatively equitable basis, force's organisations to become more efficient in producing higher quality products.

As organisations strive to become more competitive and increasingly attempt to satisfy their customer's expectations, many avenues have been explored. The current "quality revolution" sweeping the world is an excellent example of how organisations are endeavouring to improve the way in which they operate. The adoption of new technology is one avenue which manufacturing organisations amongst others, have been investigating for quite some time (Weston et al 1991). Manufacturing organisations can in fact, be increasingly considered to represent dynamic systems subject to frequently changing requirements.

The need for flexible manufacturing systems appears with consistent regularity in production literature (Gupta et al 1992, Verramani et al 1993, Chan et al 1990, Weston et al 1990, Shaw 1986, Williams 1991). In response to the increasingly dynamic nature of manufacturing organisations, manufacturing systems are being developed which integrate computers into the manufacturing process with the aim of increasing flexibility and hence gaining a competitive advantage. The integration of computers into the manufacturing environment, promises organisations the ability to reduce product life cycles, improve product quality and increase the responsiveness of an organisation to the market needs (Gupta 1988, Nagan 1992, Shaw 1987). Flexible Manufacturing Systems (FMS) can enable an organisation to increase its ability to realign its resources in accordance to market needs. The capability of a manufacturing system to respond reliably to a large mix of products being produced in small batches with minimal delay, will obviously be reflected in the organisation's productivity.

By overseas standards the average New Zealand manufacturer is medium to small in the scale of operations. Unfortunately the cost of implementing and maintaining FMS up to the present time has been quite prohibitive. This is one reason why commonly in the past it has been only the larger organisations that have invested in FMS systems. This point can be illustrated by a study conducted in West Germany from 1986 to 1988 by the Institute for Social Research Munich, (Kohler and Schmierl 1991). This survey found that the utilisation factor of computer aided production systems amongst the 1,096 companies surveyed, with the number of workers under 100 was less than 30%. Compared to almost 80% for manufacturing organisations with over 1,000 workers.

The introduction of flexibility into a manufacturing environment requires high investment, a numerically controlled turning (Gupta and Goyal 1992) centre costs (US$) 200,000 to 400,000, while the cost of an entire FMS for a medium size organisation will be many millions of dollars. A review of manufacturing software available in New Zealand (Appendix I) demonstrated that manufacturing software of even moderate capability regardless of flexibility, for a small to medium size company costs in the order of (NZ$) $250,000 and about. Because of the high cost of such manufacturing systems available within New Zealand, a need was perceived for a low cost flexible manufacturing system for the small to medium sized manufacturer.

# CHAPTER 2

## REVIEW OF LITERATURE

Table of Contents

## 2.1. Introduction

A review of literature was conducted in the area of flexible manufacturing control with the aim of gaining insight into a number of areas critical to ongoing research at the Department of Production Technology Massey University. The area of flexible manufacturing is vast, therefore investigation was confined into two major topics. Control models for flexible manufacturing environments, especially heterarchical and hybrid models. Real time scheduling within hybrid control systems including the investigation of virtual representations of physical devices.

## 2.2. Review of Current Control Models for Flexible Manufacturing Systems

### 2.2.1. Cellular Technology

During early stages of the development of FMS's, ie since the 1970s, many systems were installed without proper preparation (Verramni et al, 1993). Thus the performance and effectiveness of these systems were often disappointing. Since then an emerging technology has been cellular technology (Verramni et al 1993, Shaw 1987, Weston et al 1991, Rogers 1991, O'Grady and Lee, 1991). This is where machines or devices are grouped into cells, the devices in these cells often differ for different FMS's. One common type of grouping, is to gather devices into a cell, in which each device can perform a specific yet different task. This cell can then be set up to process a particular product family. Once the cell has been set up to process a particular family of tasks it then has a competitive (Shaw 1987, 1986) advantage over other cells, on tasks within that family.

Through organising the shop floor in a cellular fashion it is possible if the flexibility has been designed into the flexible manufacturing system, for the cells to be re configured individually to process several different tasks at once. This approach also allows several cells set up to process similar families of tasks, to transfer tasks between them. This is required in the case of a cell becoming overloaded or a fault occurring in one of the cells. A further advantage of the cellular design is that it lends itself towards modularity, which is an important characteristic of a flexible manufacturing system.

To decide which control methodology would be best employed in the design of the Flexible Manufacturing Control System (FMCS) it was important to investigate and compare existing methodologies. The two major control schemes currently in use are hierarchical and heterarchical control. These two control methodologies are radically different, one being centralised and the other distributed. To investigate the best control methodology to use in the FMCS a better understanding of the differences between the control schemes must be gained.

## 2.2.2. Hierarchical Control Structures

Most shop floor control structures are based on the hierarchical, multilevel approach. The hierarchical control model consists of multiple levels, which reduce complexity and limit responsibility. The bottom level of the hierarchy is filled by physical devices such as CNC machines. The task of these devices is to undertake actions dictated by, and to provide feed back for the next highest level. The levels in the middle layers undertake commands issued from above and issue commands to the lower level. The highest level is responsible for planning actions and receives information from the lower level then issues commands based on the calculated plan. Each level conforms to several rules defining its behaviour (Jones and Saleh 1990, Jones et al 1989); each level has a distinct planning horizon that decreases as you go down the hierarchy and control resides at the lowest possible level.

The different architectures resulting from the application of the above guidelines, differ along the following lines for hierarchical control systems;

- The number of levels and functions assigned to each level.
- The specification of control paths between supervisors and subordinates.
- The handling of data and communication.

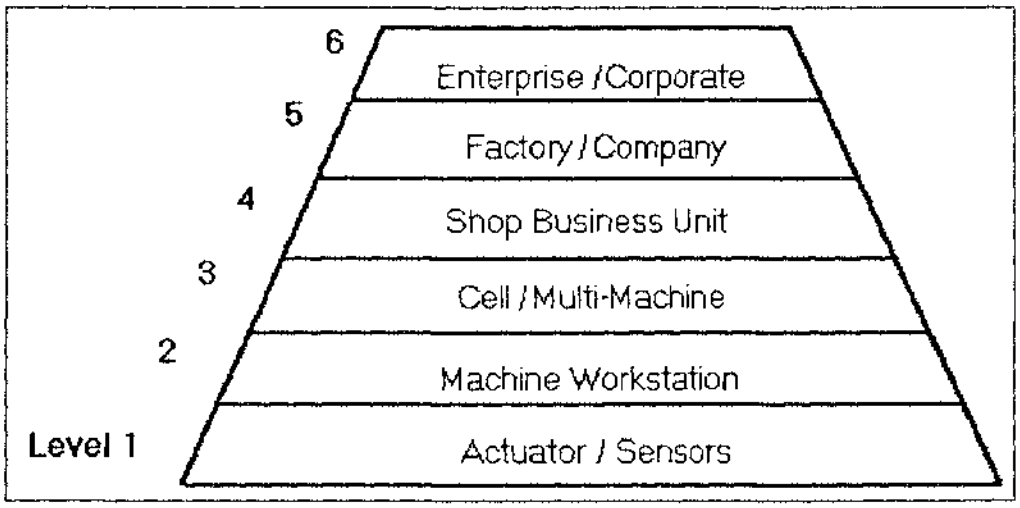



Figure 2.1
Typical factory hierarchy (Adapted from Weston et al 1991)

The model in figure 2.1 demonstrates the different levels that constitute a FMS, each of the levels has a specific purpose.

-Enterprise/corporate:

The highest order decisions are made at this level affecting the long term prospects of the company. Data required to make these decisions is typically in the order of months or possibly days, nothing compared with the seconds or milliseconds needed

for lower level decisions below. This is generally the highest level of management, traditionally this level was distant and often unaware of what actually happens down at the plant level. The information required here is usually a compiled summary of the data collected over a long period of time. Topics such as the Quality standards and corporate budgeting are often considered at this level.

-Factory/company:

At this level many of the support staffs such as the purchasing managers and the sales team are involved, usually separate from the production unit, nevertheless are required to maintain an efficient working environment. Tasks such as scheduling and departmental budgeting's are often dealt with at this stage. The integration of these people is important in a CIM system, as they provide the upper management's eyes on how the organisation is functioning.

-Shop business Unit:

At this level, line mangers decide upon the medium to long term goals of the organisation. Although time critical information is needed, the time factor is still relatively long term compared to that of the lowest levels of operation. Here the timeliness of information begins to become important as decisions made here, immediately effect the efficiency of the lower levels and the output of the factory as a whole. Decision on the maintenance schedule, down time analysis and the management reporting functions are common tasks at this level.

-Cell/multi machine:

This stage involves the highest level of critical control. Here decisions made affect the production unit almost immediately. Judgments made at this level generally for fill the production support role, such as data acquisition for quality control, the ordering of materials, the updating of buffers and the assignment of jobs.

-Machine/CNC Device:

This level actually controls what is physically being done. The decisions made effect task performance directly. Here the decisions must be made precisely and quickly. The decisions require real time data that is crucial to correctly controlling the processes at the shop floor. Control decisions are generally of the type concerning physical limitations.

-Actuator/sensors:

At this level there is no decision making nor is there any "intelligence" to offer the ability for control. Any control that exists is in the form of physical limitations. The equipment used here must be reliable as the input and outputs of the system are directly related to the ability of these devices.

An important characteristic of many hierarchical control architectures is that control flow and data flow is equated. This means that the data needed to execute a command must be transmitted along with the command. The only exchange of data allowed in this type of system is between a supervisor and 'its' subordinates. This in effect means no peer to peer communication. While this was required in the early days of point to point communications, modern technology eliminates this need (Jones and Saleh 1990)

### 2.2.3. Heterarchical Control Structures

Due to several constricting disadvantages of the hierarchical control approach, researchers (Hatvany 1985, Duffie and Piper 1986, Veeramani et al 1993, Duffie and Piper 1994) have proposed the use of heterarchical control to eliminate this deficiency. Heterarchical systems use a distributed control approach unlike a centralised system. All the entities within the system are treated as equal and no single entity has direct control over any other. There is only one level and everyone resides on it. Entities co-operate as equals through collective agreement to negotiate a schedule and the manufacturing of products. The negotiation process between the systems entities is often based upon an "auction" (Tilley and Williams 1992, Shaw 1987, Veeramani et al 1993, Jones and Saleh 1990), where the entities place bids to obtain Jobs and resources.

One way in which to best describe a heterarchical system, is that it is a system where there is no global information base that can fully describe the state of the system. Instead the information is distributed around the entities (Duffie 1991). The distributed nature of heterarchical control architectures gives them a number of important characteristics. Independence amongst the different entities means the system is modular (Jones and Saleh 1990, Veeramani et al 1993). Entities also require a relatively high level of intelligence, to enable them to autonomously make decisions.

The main reason for the introduction of this form of control architecture is added system robustness, that is if one vital communication link fails, it does not affect the whole system (Duffie 1991). The structure is thus said to be fault tolerant as there is quicker recovery from broken lines, detection and containment of faults and a reduction in system complexity. There is also the added advantage that the responses to changes are faster, as the controllers do not have to ask higher level entities for advice on what should be done. Further more the flexibility of the system is said to be very high, as horizontal expansion does not incur a vertical system re-configuration cost.

## 2.2.4. Comparisons between Heterarchical and Hierarchical Control Structures

Since heterarchical control systems were first investigated comparisons have been draw between heterarchical and hierarchical control architectures. Heterarchical and hierarchical control structures can be summarised as follows.

Table 2.1
Comparison of heterarchical and hierarchical control structures

| | **Hierarchical structure** | **Heterarchical structure** |
|---|---|---|
| Control structure | Centralised | Decentralised |
| Scheduling control | Task allocation from master controller | Opportunistic scheduling |
| Scheduling Mechanism | Command passing, state feedback | Auction |
| Information database | Global database | Distributed database |
| Susceptibility to failure | High | Low (failures are localised) |
| Communication | Vertical communication | Message passing between entities |
| Flexibility | Low, system requires considerable re-engineering to expand | High, modular system provides high adaptability |
| Complexity of design | High level of complexity as system size increases | Less susceptible to increase in complexity as systems size increases |
| Use of global information | High, all decision are made with full knowledge of state of system | Low, decision are based on the summation of local information |
| Application | High, almost all shop floor controllers | Low, Most systems are at a research/simulation stage |

The comparison shown in table 2.1, details the major advantages and disadvantages of both approaches, however it fails to outline some of the more subtle points. Thus it is felt by some (Jones and Saleh 1990) that "while a heterarchical architecture is well suited for modelling the process of designing a 'manufacturable part', it is not well suited for shopfloor control." In the design problem, there is a clear goal but there is no clear supervision. Moreover, free data exchange and negotiation is imperative. However we believe that this approach (heterarchical) is not well suited for the highly structured, unpredictable nature of shop floor activities in which data, raw materials, tools and fixtures are needed to carry out assigned tasks."

There is however strong supporting evidence in favour of heterarchical control structures in the form of research done at the University of Wisconsin-Madision with the aim of evaluating hieratical and non hieratical control systems, (Duffie and Piper,

1986). Here several systems where developed side by side and the following results were obtained.

Table 2.2
Controller cost and performance comparison
(Adapted from Duffie 1991)

| | **Hierarchical controller** | **Non-hierarchical controller** |
|---|---|---|
| Lines of source code | 2450 | 1235 |
| Software development cost ($US)[a] | $61,250 | $30,875 |
| Expansion software cost ($US)[b] | $960 | $0 |
| Machine utilisation[c] | 64% | 60% |
| Complexity | Highest | Lowest |
| Flexibility | High | Highest |
| Modifibility | High | Highest |
| Fault tolerance | Low | Highest |
| Intelligent parts | No | Yes |

[a] At $25 per line of source code.
[b] 24 hours per machine added at $40 per hour.
[c] From simulated fault free cell operation, random part mix.

The above table (table 2.2) combined with the previous comparison between the two control architectures demonstrates that although heterarchical control systems appear promising they still have certain disadvantages. In fact both methodologies have strengths that complement each other. IT has been suggested that a superior control system can be constructed, by not choosing one or other of the two control methodologies but in fact to attempt to combine the advantages of both systems. This approach is not altogether new, as limited research has so far been conducted into this area. One investigation (Jones and Saleh 1990) into this Hybrid (Rogers 1991) approach has proposed a control architecture that combines the strengths of both hieratical and heterarchical systems.

The proposed architecture has the following characteristics:

- It separates control, data management and communications management.
- Distributes both decision making and control.
- Requires each module to work on several jobs simultaneously.
- Provides for limited negotiations between a supervisor and its subordinates.

In the case of a cellular approach taken in a heterarchical control system, the scheduling (Shaw 1987) tends to behave as network of queues where tasks arrive into the system dynamically over time. The cells are able to communicate amongst themselves over a Local Area Network (LAN), for the purposes of negotiation.

## 2.3. The Hybrid Control System and Real Time Scheduling

Several approaches have been proposed for the scheduling of manufacturing systems, mathematical programming, distributed dispatching rules, simulated annealing and heuristic's (Maimon and Gershwin 1988, Kochhar and Morris 1987, Perkins and Kumar 1989). In order for a scheduling approach to be practicable, such approaches require real time scheduling. This has the result of reducing scheduling lead time. Traditional manufacturing systems most commonly utilise a complex schedular on a centralised computer, this limits the ability of the system to achieve real time scheduling. It is not unusual for a schedule to take several hours to generate. Unfortunately a manufacturing system is not stable and is continuously changing, thereby making a schedule with a significant lead time, out of date by the time it is completed.

Centralised scheduling tends to operate through using problem decomposition, this is where decisions are made at the top and broken up into workable solutions, until a level is reached where the solution can be preformed physically. This sort of scheduling however is not applicable to a hybrid or a heterarchical control system. Both of these control architectures have no centralised processor and thus no place to generate a centralised schedule. Secondly the concept of a centralised schedule violates a number of fundamental principals concerning the operation of heterarchical and hybrid control systems, this is because the actions of the entities within the system would be dictated by the schedule.

A number of methods have been proposed which attempt to undertake real time or near real time scheduling of a hybrid or heterarchical control system. These methods are all based upon a negotiation process (Duffie 1987, Veeramani et al 1993, Shaw 1987, Jones and Saleh 1990). Whilst all these methods use a negotiation process, the actual functionally of the negotiation varies between research groups. Three different types have currently been investigated, auction or bidding approach (Veeramani et al 1993, Shaw 1987, 1986, Tilley and Willams 1992), local simulation (Duffie and Prabhu 1994), Multi level/multi layer scheduling (Jones and Saleh 1990).

The bidding approach operates by allowing all entities within the factory to place a bid for a particular resource that requires scheduling. The bid placed by each entity within the system is based upon detailed local information about the entity, and describes its present state and need for the resource. The resource is then allocated in accordance to a number of rules used to consider all bids placed by entities.

Research around the world (Shaw 1987, 1988, Tilley and Willams 1992, Veeramani et al 1993, Bakker 1988) has shown that a bidding approach is a feasible method of distributing jobs throughout a manufacturing system. One study (Shaw 1987) was conducted in order to evaluate the performance of a network wide bidding system as a dynamic scheduling algorithm. A primary objective of the simulation based study was to compare the performance of the bidding algorithm with other scheduling methods. The results of the research demonstrated that the bidding algorithms performed better in terms of mean flow-time, in-process waiting time, late jobs and tardiness than the centralised myopic scheduling methods used.

As discussed by (Shaw 1987) such an approach essentially treats the scheduling problem by a multi agent problem-solving paradigm: because the whole scheduling problem is too complicated, the set of problem solving agents "the cells" carry out the tasks collectively. Just as in human organisations, bidding is employed as a mechanism for co-ordinating the execution of tasks, in this case amongst cells. This paradigm was developed by research in artificial intelligence (Davis and Smith 1983, Shaw 1984) and has been applied to various types of distributed systems such as sensor network (Smith 1980) or computer networks (Malone et al 1983, Ramamritham and Stankovic 1984).

Since the pioneering work of Lewis (Lewis 1981), research into auction or negotiation based shop floor control system has continued at several research institutions. Several different variations on the previously mentioned auction based methodologies were investigated. One such investigation (Shaw 1987) proposed a distributed control scheme where dynamic system information is managed by a distributed database system. Each cell maintains its own local world model, while systematically co-ordinating with other cells through task sharing and bidding. By negating the need to collect dynamically changing information in a global database, certain communications overheads are avoided. One approach proposed was a two level scheduling method (Duffie 1991), where one level involved the distribution of the task between cells, the second level focused upon distributing the task amongst the machines within the cell. Shaws approach utilised a group technology model, differed from other single level auction based scheduling systems, similar to that investigated by Veeramani et al (Veeramani 1991), which were proposed for the shop floor distribution of tasks throughout a FMS.

The local simulation approach proposed by Duffie (Duffie 1994) is a scheduling method that utilised the distributed architecture of a heterarchical control system to generate a host of simulations to undertake the scheduling of resources. Each entity considers only its local information when generating a tentative local schedule, which is referred as a local plan. While this allows plans to be generated in a highly distributed manner, entities can generate conflicting plans, and no explicit effort is made to prevent negative interactions between entities. All local plans are evaluated together so that entities can obtain feedback about the global worth of their plan and use it to select the best local plans. Plans are continually generate and evaluated by the entities, when a better schedule is found the system is re-scheduled.

Multi level/multi layer scheduling within a hybrid control system is a complicate scheduling method that mixes traditional scheduling techniques with simulation. This method proposed by Jones (Jones and Saleh 1990) uses a basic traditional schedule generated by a single higher level controller. This schedules part families amongst groups of machines, cells. Due to the simplicity of the centralised schedule it is stated that the computational requirements are minor. Once this first level of scheduling has been undertaken, the individual parts within the part families are scheduled amongst the available machines through the generation and evaluation of simulations at the cell level.

## 2.4. Virtual Manufacturing Devices

The concept of a Virtual Manufacturing Device (VMD) is not new (Brill and Gramm 1991), previously the use of the term VMD appears to have centred on the Virtual Manufacturing Device defined in the Manufacturing Messaging Specification (MMS) (Brill and Gramm 1991, Caven and Jackman 1990, Weston et al 1991). MMS is an International Standards Organisation/Open Systems Interconnection (ISO/OSI) application layer (Layer 7) protocol used in the Manufacturing Automation Protocol (MAP) to support communications between programmable devices in CIM and process control environments. MMS defines the content and order of messages between network stations to establish standardised communications between programmable devices (Caven and Jackman 1990). MMS is a client server will protocol.

MMS uses the VMD as an abstraction of a real programmable device. Its VMD is an abstract representation of the common characteristics of all real manufacturing devices and represents externally, the visible behaviour relevant to MMS. This makes MMS a standard that allows MEs to communicate, as each ME has a standard interface. MMS only describes the effects of the services on the VMD and does not prescribe a specific implementation or transformation to a real function and is not yet widely available for all MEs (Weston 1991).

The concept of using virtual representations of machines has been used in other research, whilst not being called VMD's, they are effectively the same. Duffie and Prabhu (Duffie and Prabhu 1994) uses a virtual representation of a machine as a basis for simulations conducted at the machine level. These virtual representations exist along side the real system. This allows the simulation to interact with the virtual system in much the same way as it would with the real system.

In general, virtual models of a real system are used extensively in research concerning hierarchical and heterarchical systems, (Weston et al 1991, Duffie and Prabhu 1994, Caven and Jackman 1990, Brill and Gramm 1991, Dakroury and Elloy 1988). These models are used to simulate the state of a physical device in software, so as to either decouple the control of a device from its implementation or as a method of standardising the interaction between a manufacturing system and a physical device (Dackroury and Elloy 1988).

# CHAPTER 3

## CONTROL MODEL FOR A FLEXIBLE MANUFACTURING SYSTEM

Table of Contents

## 3.1. Introduction

Once the focus and scope of the research had been defined and a course of action decided upon, an appropriate control model was required to be formulated which would imbue the desirable characteristics of the design phase into the implementation of the control system. A control model was developed, based upon a combination of previous research and several novel design concepts. The model used a combination of both heterarchical and hierarchical control methodologies, called a hybrid control methodology. This model consisted of three separate layers, factory, cell and low level device control or Intelligent Entity, each layer having a defined purpose.

The design and operation of each of the three layers of the control model was segmented into functional entities and are discussed both in isolation and in relation to other entities. The various functions which each section consist of should not be treated as a conclusive list of all functions required, the author accepts numerous other functions exist and will be required as the FMS control system becomes more advanced.

## 3.2. Development of the Hybrid Control Model

Before the development of the hybrid control model could begin with it was important to detail the design considerations to be taken into account. These considerations were created in order to limit the scope of the design and thus make the implementation of the model more feasible. A number of characteristics were also defined to which the model must conform. These characteristics realise certain factors which have been deemed important to the research in the Department of Production Technology. Such as to utilise a hybrid control architecture and to enable the model to be directly applicable to the New Zealand manufacturing environment.

### 3.2.1. Scope of Proposed Hybrid Control of a Flexible Manufacturing System

The combination of design considerations and the model characteristics define the limits or scope of the model design. By strictly defining the scope of the model design it was possible to increase the probability that the implementation of the model would be successful.

Several characteristics were developed (Jones and Saleh 1990, Golenko-Ginzburg and Sinuary-Stern 1988, Weston et al 1991) as a guideline for the design and implementation of the hybrid FMS control model.

Design Characteristics

1. The decision making ability at the cell control level will be distributed.

2. The integration of humans and machines into the control system should be seamless and interchangeable.

3. No master slave relationships shall occur at the cell control level.

4. Each entity shall have autonomy.

5. Flexibility and re-configurability should be maximised.

6. The individual entities at the cell control level shall be governed by rules and driven by goals.

7. The controlling ability of the cell controller will be generic so as to enable interfacing with a wide variety of different low level devices.

8. Faults should be contained locally, the recovery from faults should be controlled locally.

9. No entity should assume the co operation of another entity.

10. The auction methodology will be utilised as an opportunistic real time schedular.

11. The model should have an applicability to a wide variety of operations and organisations.

12. The model be based on a hybrid control philosophy.

In addition to these design characteristics a number of considerations were also required to be taken into account which limited the scope of the design.

Design Considerations

1. The implementation will be based upon readily available technology. This limited the scope of the design as it required the model be able to be implemented upon existing resources and in available technology.

2. The implementation requirements must be modest in terms of financial resources. The design was limited somewhat by the availability of potential resources. Thus the design must be feasibly implemented upon either existing resources or those resources that can be reasonably obtained.

3. The implementation of the model will focus on the feasibility of utilising an existing SCADA computer package owned by the Department of Production Technology.

This SCADA package called FIX DMACS will be used as extensively as possible, to access its abilities as a tool for the implementation of the FMS control model.

### 3.2.2. The Hybrid Control Methodology

In order for the control structure to be hybrid in nature it must demonstrate a combination of both heterarchical and hierarchical control. It has been proposed that the lower levels of control (in terms of a hierarchical model), ie the cell controller will operate a heterarchical nature whilst the upper levels will operate as a hierarchy. It is the cell control or shop floor control which lends itself to the heterarchical approach. This is because;

1. Heterarchical control architectures promise high reliability with superior performance.

2. Cells often are required to operate over a large physically distributed area.

3. The factory floor is required to be flexible, reconfigurations and random disruptions are both common place events.

The auction process allows the distribution of tasks within the system according to the decision criteria utilised in the auction. Most of the research conducted into heterarchical control architectures has focused on the shop floor control level (Tilley and Williams 1992, Duffie 1991, Veeramani et al 1993, Bakker 1988). A number of cell controllers linked together on a LAN with each cell controller having jurisdiction over the devices within the cell, is an ideal environment to operate a heterarchical style control architecture.

Therefore a division of the different levels of a FMS control system into a mixture of both hierarchical and heterarchical has been proposed, as detailed in figure 3.1.

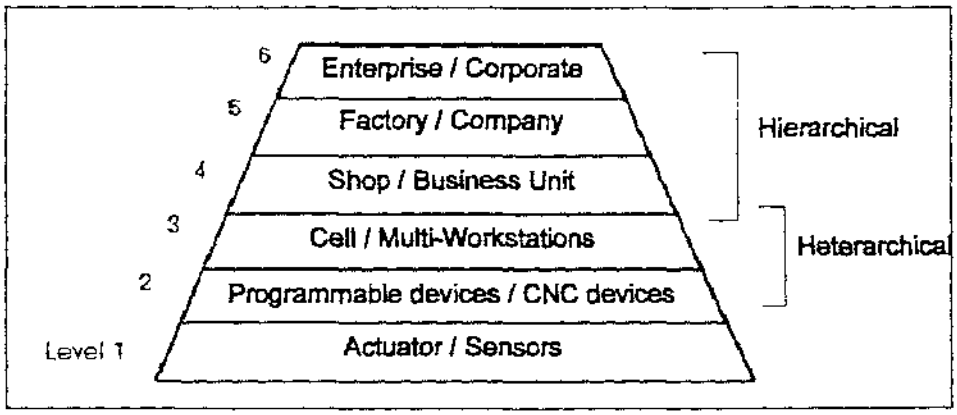



Figure 3.1
Factory operational model in a hybrid methodology
(Adapted from Weston et al 1991)

Figure 3.1 shows the levels directly below the shop/business unit level were developed with the heterarchical control architecture in mind.

### 3.2.3. Proposed Physical Model of a Hybrid FMS Controller

With the design characteristics in mind, a physical model was proposed. The model consists of three layers, based upon traditional/hierarchical lines, with the major constituents being a Factory controller, Cell controller and the Intelligent Entity.

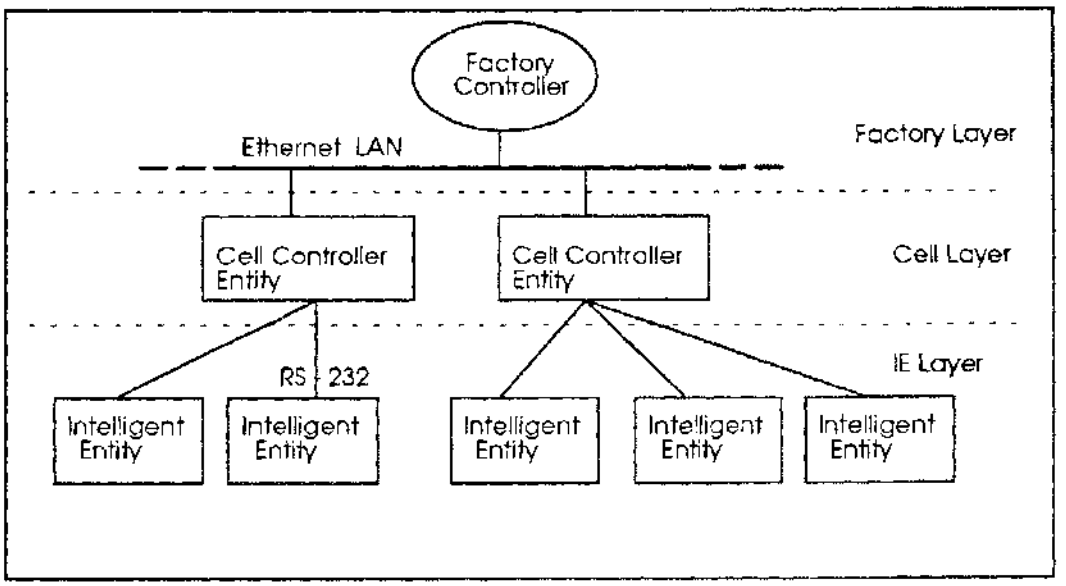



Figure 3.2
Three layer functional control model

Figure 3.2 depicts the physical control model developed, this model shows the physical layout of the major control entities in the model. The factory controller rests at the top of the factory hierarchy and is responsible for the master scheduling and for the definition of the structure, content and work of the factory. Each of the cell controllers are of equal importance and lie at the next level down. One major aspect of the uniqueness of this control model is expressed at the boundary between the cell and factory controllers. The manner in which jobs are distributed amongst the factory controller is deterministic. The cell controllers negotiate amongst themselves for the right to process a particular job, this process of negotiation is resided over by the factory controller, who decides which of the cell controller is most capable of undertaking the job. Thus there is no overall plan of product flow, the "scheduling" should in fact be conducted as required and in near real time.

Each cell controller has jurisdiction over a number of IE's and is responsible for the control and monitoring of its IE's. At the cell control level a second unique aspect to the model is best expressed. Here the integration of people and machines is hoped to be achieved. Therefore it was important to design the cell controller - IE interface very carefully, so it was in-fact possible for both humans, automated devices and robots to interact which the cell controller. The Intelligent Entity consisting of one or more programmable devices combined with a physical machine such as a CNC Lathe or Mill. The IE enables each individual CNC device with separate and incompatible controllers to be directed from the Cell controller through a standard interface (Weston et al 1991). The IE's have an equal status amongst themselves, thus no one IE will have priority over any other. When considered, an IE actually consists of several smaller or primitive Manufacturing Entities (ME's), the programmable device controlling the machine, the factory installed CNC controller and the CNC device itself.

The communication between the factory controller and the cell controllers is done via the LAN. Having a bus LAN for all the cell controllers to communicate over with equal priority, is important in integrating the heterarchical aspects of the control in with the hierarchical. The communication between the IE's and the cell controller is hierarchical in nature and therefore need only to be point to point, as no communication need take place between the IE's themselves.

### 3.2.4. Basic Definitions and Assumptions for Formulation of the Model

Some basic assumptions were made at the early stages in the formulation of the hybrid FMS model, these assumptions were made to provide a reference of terms so as to provide consistency through out the project. Further more the way in which terms such as job's and cell's are defined effect the physical implementation of the system. A list of definitions and accompanying assumptions are shown in section 5.2.1.

## 3.3. Factory Controller

The factory controller as mentioned previously sits at the top of the hierarchy and provides management of key aspects of the factory. The factory controller does not have direct control over the cell control layer, no functions of the factory controller are required to direct every action of the individual devices at the lowest level. In respect to the scheduling of jobs within the factory this means that the factory controller provides only a master schedule of when jobs require processing and what resources are required. The actual inter-cell scheduling is carried out by an auction process, which takes place between the cells in the factory.

The functions which the factory controller undertake are those indicated in figure 3.1 above the cell/multi-machine level (Weston et al 1991, Rogers 1991). These functions performed at the higher levels of a FMS, such as manufacturing engineering functions, information management functions, resource planing, tooling, factory configuration, factory monitoring, job definition and factory simulation. These tasks are represented in figure 3.3 as a functional components of the factory controller.

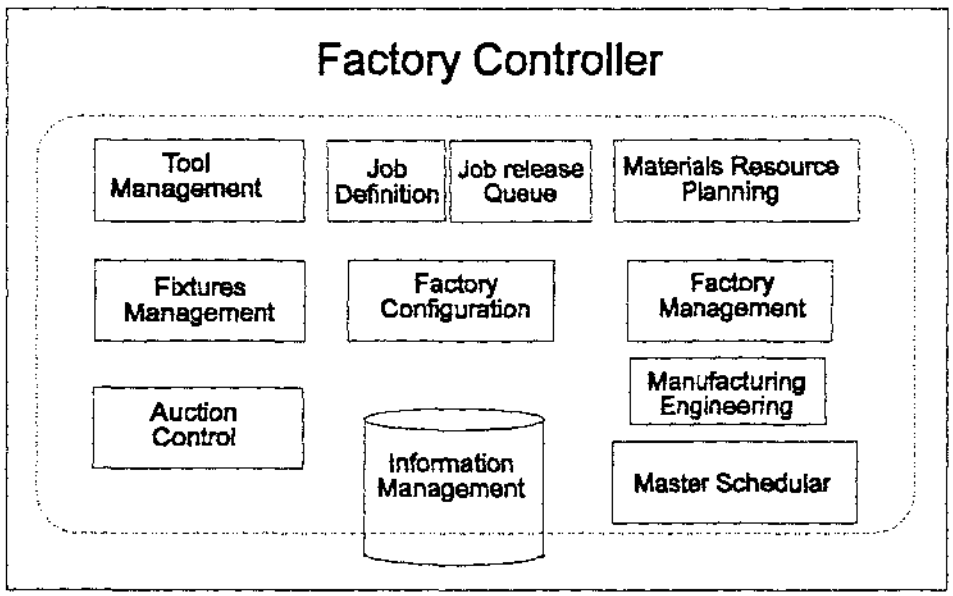



Figure 3.3
Factory controller functional model

### 3.3.1. Factory Control Entities

1. Fixtures Management. The Fixtures management entity is the entity which manages fixtures resource, this entity must interact with the Master Schedular entity which schedules resources for all the cells within the factory.

2. Tool Management. This is the entity responsible for the management of the tooling resource. The tools available in the factory must be monitored and allocated as required.

3. Job Definition. The job definition entity is utilised by the operator to define new jobs to be manufactured by the factory. This function also requires the operator to use the manufacturing management functions to design jobs as required.

4. Job Release Queue. The Job Release Queue is the entity responsible for releasing new jobs into the system to begin manufacturing. This is done in accordance to when the job is required, which is decided at the job definition entity, by the operator.

5. Factory Configuration. Factory configuration is the function which enables the operator to enact a software base change of the factory representation. This also enables the user to define and expand the software model of the factory.

6. Factory Management. This function is concerned with the monitoring of the operation of the factory. This enables an operator from one point to gain access to summarised real time and historical information concerning the operation of the factory.

7. Manufacturing Engineering. The design and creation of machine files is handled by this entity. The CAD/CAM functions of this entity are important operations in the creation of new jobs.

8. Master Schedular. The Master Schedular manages the consumable resources of the factory to ensure future jobs will have the resources to be completed. However the scheduling of resources within a factory were there is no defined process flow nor set job schedule is a difficult problem.

9. Materials Resource Planning. This entity operates in combination with the master schedular to plan the usage of resources to enable the factory to operate as planned and to maintain the processing of jobs in the time frame specified.

10. Information Management. This entity at the factory level manages the information resources, such as machine files, job files factory layout information. This requires the organisation and transportation of appropriate information to the destination cell.

11. Auction Control. The Auction Control entity is the entity which acts as the decision maker in the auctioning of resources and jobs within the factory. Whenever a job is required to be allocated the job release queue releases the job into the system and a negotiation or auction process is initiated. The Auction control entity decides upon which cell will "win" the job for processing by comparing the "bids" lodged by each capable cell, then selecting the most appropriate one.

### 3.3.2. Factory Controller as an Integration Tool

One goal in the design of the factory controller was to enable it to act as a factory integration tool, combining many different functions into a single service. When formulating the model for the factory control it was recognised that there would be tremendous advantages in being able to integrate a variety of different applications each undertaking different functions of the factory controller. Through using a number of-off-the shelf applications combined together, it was envisaged that major portions of the factory controller could be implemented with minimum effort. It does however introduce the problem of having to create a platform which will enable the integration of a variety of different applications, such as CAD/CAM (Nahavandi and Preece 1993).

It was decided it would be best to use a number of applications which operate using a similar user interface, thus giving the user consistency. These applications must be able to communicate with other applications operating within the same environment in order for information to be passed amongst them. Obviously no combination of off the shelf applications would be capable of providing all the functions required of the factory controller. Thus a number of custom applications were required to be written. These applications were required to operate using a similar interface as the other applications with which they co-existed. This language would be used to undertake the tasks which no other application within the FMS control system is able to do. The major task of this custom application would be to "glue" (Figure 3.4) all of the other applications together and provide the necessary information and communications management between them. This custom application would be the "window" to which the user would gain access to the FMS control system.

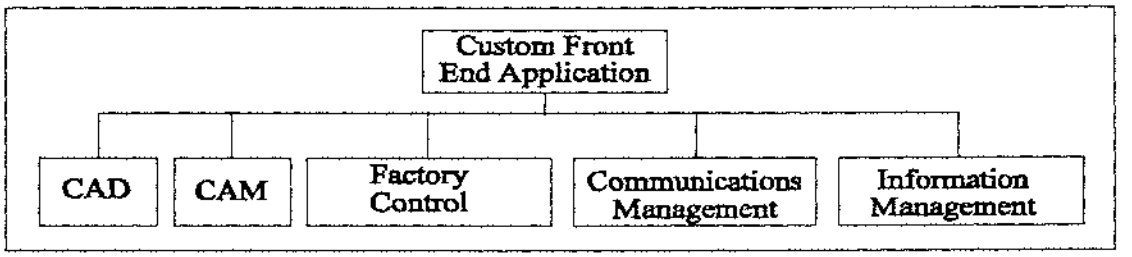



Figure 3.4
Layout of factory controller application software

## 3.4. Cell Controller

The cell controller, like the factory controller consists of a number of ME's which are responsible for directing the actions of the IEs. The cell controller is responsible for obtaining tasks and resources for each of the IEs under its control. This entails dispatching tasks to the IEs and then monitoring their progress.

A typical FMS cell has three to four machines (Chan and Bedworth 1990) including inspection and internal materials handling, this is accomplished either by a automated device or a human operator. Although the FMS system is designed to be flexible, once the cell has been set up to process that particular part family, it will hold a competitive advantage over other cells in relation to one particular part family. Each of the machines in the cell will be of a different type, with a maximum number of four devices in any one cell. The cells are able to communicate amongst themselves and with the factory controller via the LAN. This will enable the cell controllers to conduct an auction in conjunction with the factory controller.

The nature of the cell controller as a whole is to co-ordinate the operation of the low level devices rather than to dictate their every move (Rogers 1991). The aim is not to decouple the low level controller completely from the cell controller, but to give the low level device enough "Intelligence" to use the detailed real time information it has available, to take care of the detailed implementation of the decisions passed down from the cell controller. This enables the processing requirements to be distributed and also reduces the communications requirements between the low level device and the cell controller.

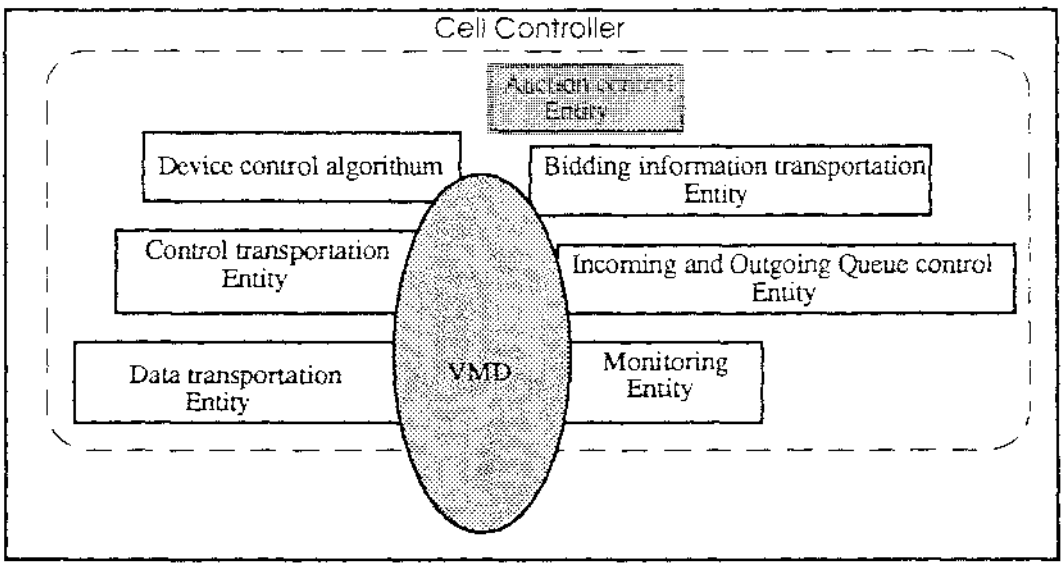



Figure 3.5
Cell controller functional model

Figure 3.5 details the major functional components of the cell controller, the relation of the components to the Virtual Manufacturing Device is also shown.

### 3.4.1. Cell Controller Entities

1. Data transportation entity. The data transportation entity is responsible for the transportation of data files from the central file store to the destination cell controller. This involves retrieving the data file from the file store via the LAN and then down loading the correct information into the appropriate cell.

2. Control transportation entity. Control transportation is the task of transporting the control information from the cell controller to the IE and gathering feedback information. This task interfaces with the cell controller, from which all the control information must come and the IE. The gathering of the feedback information from the IE is a very important task which enables the control algorithm for the particular IE to close control loop.

3. Device control algorithm. The device control algorithm is the generic algorithm which controls the IE directly. This algorithm issues commands based on an evaluation of the IE's current state and monitors the IE via the control transportation entity to see how the commands are being carried out. The algorithm has distinct stages of operation which represent the different phases of the operation currently being performed at the IE.

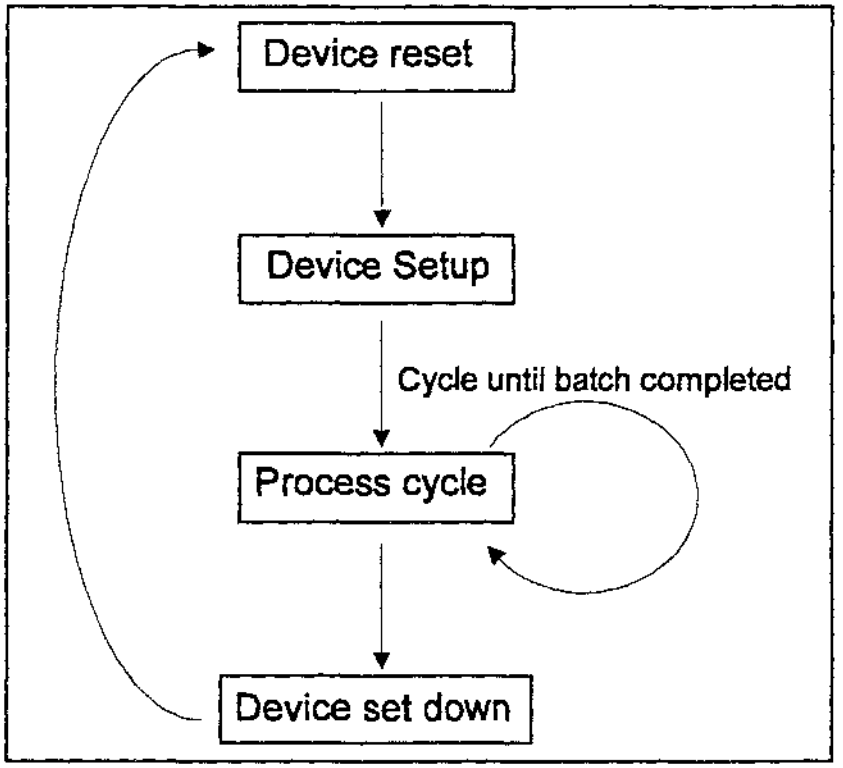



Figure 3.6
Device/machine control algorithm

Figure 3.6 represents in simplistic terms the different stages of the algorithm which controls an IE. In each cell there must be one control algorithm for each machine/IE which is to be controlled. By making the algorithm itself generic, a single algorithm is

able to control any type of low level device so long as the control transportation entity is able to transport the commands to and get feedback from the IE.

The generic nature of the control algorithm is an important part of the flexibility of the FMS control system. This generic architecture is achieved through using a common set of services, similar to MMS via the Virtual Manufacturing Device (VMD, see section 5.2.1.) to communicate between the IE and the cell controller. It is the device control algorithm which uses these services to direct the actions of the low level device.

4. Monitoring entity. The monitoring entity provides the user with a visual representation of the current state of devices being controlled at the cell. The monitoring entity does this by interrogating the VMD, which reflects the current state of the low level device. The monitoring entity can then display relevant information about the current state of the low level device to the user/operator. This enables humans to be integrated into the FMS system, as this system can allow the user to monitor and effect the state of the VMD and thus affect the state of the low level device.

The monitoring entity can also report summarised information up to the higher levels, thus the management would be able to obtain simplified information about the state of any cell in the factory.

5. Incoming and outgoing queue control entity. This entity controls the loading and unloading of the buffers or queues for each CNC device in each cell. These buffers store task information, ie how many parts to produce, what task to process next. As jobs are dispatched into the system, tasks are allocated by the process of negotiation to individual devices within the cells. Thus it is important for a device to be able to load its buffers with new/incoming and completed/outgoing task, so as the multiple jobs can be released into the system at any one time. An incoming queue will be loaded with a task when the appropriate device "wins" an auction, the out going queue will be loaded with a task when a device completes a task.

6. Bidding information transportation entity. This entity is responsible for the transportation of the bidding information between the originator of the auction and the cells which will be taking part in the auction. This entity must provide timely transportation of the bidding information so as to enable the auction to conclude within a reasonable time frame.

7. Auction control entity. This entity is responsible for the initiation of the auction when it is appropriate. This occurs when devices within the cell controller reach certain states, at this point it is necessary to hold an auction in order to determine which cell within the factory will "win" the right to process the job at the centre of the auction.

## 3.5. Intelligent Entity

A manufacturing entity is "what ever is required to accomplish one or more manufacturing activities" (Weston et al 1991). The devices at the lowest level, such as a CNC lathe, fit within the definition of an ME. However, it was decided to introduce an extra layer of control "Intelligence" on top of the low level device. An intelligent entity (IE) consists of two layers, a programmable controller connected to a cell controller and a CNC device controlled by the aforementioned programmable device.

This next highest level of control was introduced so as to enable a generic interface between the cell controller and the low level device. This generic interface allows any type of low level device to support a standard array of services with the addition of the extra layer of control (Jones and Saleh 1990). This multi layer control approach to the support of various multi vendor low level devices, enables a cell controller to support a wide variety of low level devices allowing the FMS control system to maintain flexibility across a wide variety of end applications.

By introducing a second layer of control some of the decision making processes traditionally made at the cell control entity or the cell control layer can be undertaken at the IE. Furthermore by lowering the level of decision making and increasing the overall flexibility it is possible to better utilise the communication bandwidth available for more timely transportation of state information and free up processing time at the cell controller for other tasks.

The IE includes the low level device, the proprietary controller which may come with the low level device and the low level device itself. Together these MEs, appear from outside as a single entity, an IE. The interface to this entity is a series of data objects or groups of data that constitute the VMD. The data objects are the services that are provided by the VMD, such as START, STOP, LOAD_MACHINE, etc. The cell controller can effect the operation of the low level device by manipulating these objects within the VMD.

An IE can be defined by the following definition "What ever is required to make decisions so as to accomplish one or more manufacturing activities, which reside at the bottom of the three layered FMS control model.". This definition will remain true so long as the entity conforms to the rules governing the behaviour of an IE. These rules are given in appendix O.

### 3.5.1. Intelligent Entity Control Model

The physical components of an IE consist of two separate entities. These two entities together perform a number of different control tasks. However, the control tasks often span more than one of the physical component of the IE. Thus when viewing an IE control model, one cannot use a physical basis for separating the control functions. The Intelligent entity can be logically divided up into two distinct layers of control, these layers can then be implemented with tight coupling to give good response times

but will still be functionally separate. These layers being; generic decision rules and the specific implementation routines. This model of the IE is shown in figure 3.7.

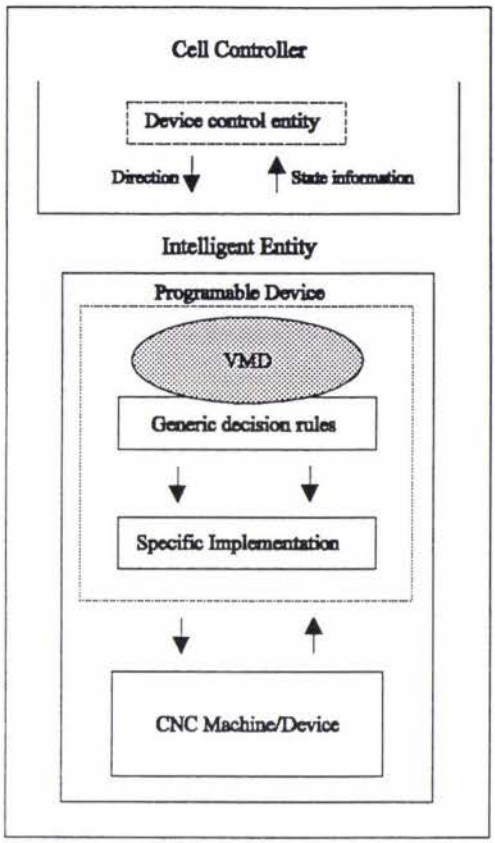



Figure 3.7
Model of IE structure

At the top of the control model is the VMD, the VMD is the interface between the cell controller and the IE. Just below the VMD layer rests the generic decisions and communications routines. These rules allow the IE to make decisions based on current information from the low level device and on direction given from the cell control layer. The IE will attempt to reflect the current state of the low level device upon the VMD. The direction given from the cell control layer will be based upon the cell controllers knowledge of the state of the low level device. The directions passed down from the cell controller represent general, generic directions on what actions the entity should take, the specific implementation of these directions is left up to the IE. The decision making capability at the IE level is limited compared to the cell control level and will mainly deal will exceptions to the normal course of events, ie error conditions, or other such events in which the greater extent of information available to the IE will allow a 'better' decision to be made at the IE level.

The control structure within the IE can be seen as a decomposition of the decision making process (Rogers 1991, Weston et al 1991). The generic decision rules take the direction based upon the state of the VMD, then make a decision based again upon the sate of the VMD. The decision is then decomposed into its functional

components, these components are acted upon by the specific implementation routines. This process is shown in figure 3.8.

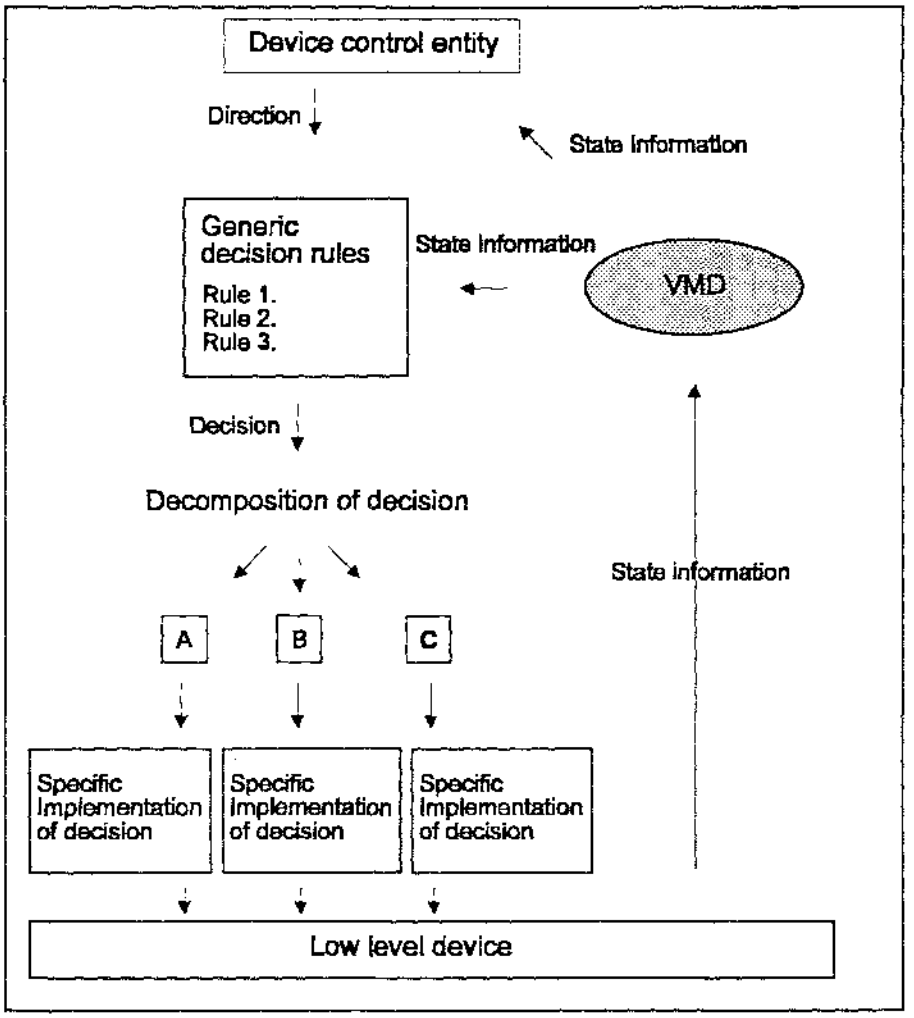



Figure 3.8
Decomposition of cell directions

The first layer (VMD) receives the message, then processes it and makes any decisions based upon that information. The second layer implements those decisions within the particular low level device which the IE is controlling.

The decomposition of the generic directions into generic instructions is performed so as the generic components of the IE can be utilised at all IEs. The specific implementation routines are the only components which will require customising for different low level devices. The decomposition into generic instructions can be conceptualised as a decomposition into primitive operations, (Caven and Jackman 1990, Brill and Gramm 1991, Jones and Saleh 1990) it is the primitives which are the custom or implementation specific executions.

## 3.6. Virtual Manufacturing Device

The concept of utilising a VMD within a FMS to provide standard interfaces between the cell controller and IE was determined as an excellent way in which to provide flexibility and modularity between the clearly defined components of the FMS cell controller. A simplified model of a VMD (Weston et al 1991) was developed that provided a host of services similar to Manufacturing Messaging Specification (MMS) services. The new model did not required all of the complexity of the MMS services. Thus, the services provided and the complexity of the message system was in comparison much reduced. The model of the VMD for this FMS control system was defined as "an abstract representation of the common characteristics of the low level manufacturing devices and represent externally the behaviour and state of that device".

The VMD allows the cell controller to manipulate and view a representation of the physical device. A core contingent of services are supported, which are generic and apply to all devices. Exceptional services are provided in cases where the core services are inadequate. This makes the VMD a modular component which can be utilised by all IEs without the need for modification.

The VMD only applies to the communications layer between the cell controller and the IE. Figure 3.9 gives a diagrammatic representation of the difference between the two different types of communication from the cell controller, ie data objects and VMDs, (which consists of a group of data objects).

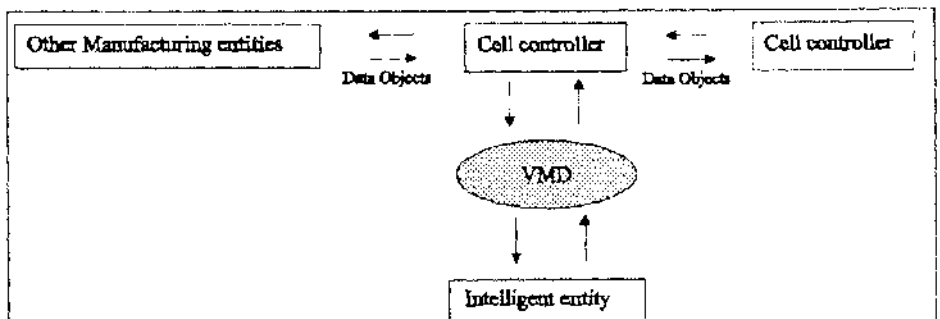



Figure 3.9
Communications VMD or data objects

## 3.7. Auctioning

The hybrid control nature of the FMS control system is reflected at the cell control level, where all cell controllers have a semi heterarchical relationship with the factory controller. This hybrid control schema utilises a series of auctions which take an opportunistic approach to the distribution of jobs throughout the factory. Rather than the traditional schedule being formulated at the factory level and then being dictated, cells negotiate with one another to determine the allocation of the job.

### 3.7.1. Definitions of Terms

Before further discussing the auction method of scheduling it is important to first define several important terms in order to maintain consistency. These definitions can be found in section 5.2.1.

### 3.7.2. The Principal Behind the Auction System

The fundamental idea behind an auction based job allocation system, is that an auction is called for by a cell who has a task which requires an operation to be performed, that cell then becomes the auction originator. This cell broadcasts over the network to all interested cells that an auction is to take place. Then each interested cell receives a group of bidding objects which provide that cell with information about the task to be performed. Each interested cell then makes a decision about wether or not it should take part in the auction. If it decides to, it then uses its detailed knowledge of its local state in conjunction with the bidding objects to calculate a bid. This bid is a "price" or "cost" for what its respective cell will take to undertake the task at hand. These bids are then transported to the factory controller which has the duty of making a decision on which cell will be allocated the task.

### 3.7.3. The Multi / Single Agent Approach

As discussed by (Shaw 1987, 1994) such an approach essentially treats the scheduling problem by a multi agent problem-solving paradigm: because the whole scheduling problem is too complicated, the set of problem solving agents "the cell controllers" carry out the tasks collectively. Just as in human organisations, bidding is employed as a mechanism for co-ordinating the execution of tasks among the cells.

Shaw's approach utilised a group technology model, whilst the proposed FMS control system model is based upon a cellular approach where only one of each type of machine was utilised in any one cell. Thus only the first level of scheduling in Shaw's (Shaw 1987) investigation applied. A single level auction based scheduling system similar to that investigated by Veeramani (Veeramani et al 1993) was proposed as the basis for the shop floor distribution of tasks throughout a FMS.

### 3.7.4. The Proposed Auction System

In the auction based system, the task requiring processing is treated by a cell as a resource. Each cell is required to negotiate for all resources, thus not only is a cell required to enter an auction in the attempt to obtain a task, it must also enter into an auction to obtain the resources to undertake the operation of that task. Therefore a cell that is bidding for a particular task must enter into an auction for tools, work pieces and transportation to complete that task.

One important point that became apparent during the formulation of this model, was that the performance of the bidding system would be greatly limited by the quality and availability of the information. Shaw (Shaw 1987) determined, as a result of the simulation research, that having more up-to-date information resulted in significantly better performance for the auction based systems. It was felt that having timely and accurate information would be essential to the effectiveness of the auction system. Unfortunately the greater the accuracy of the auction process the higher computational and communication (Tilly and Williams 1992) resource intensive the process would be become. Therefore it was decided in the design of the auction based system, that there must be a trade-off between the quality of the bid placed by a cell and the time taken for that cell to formulate the bid.

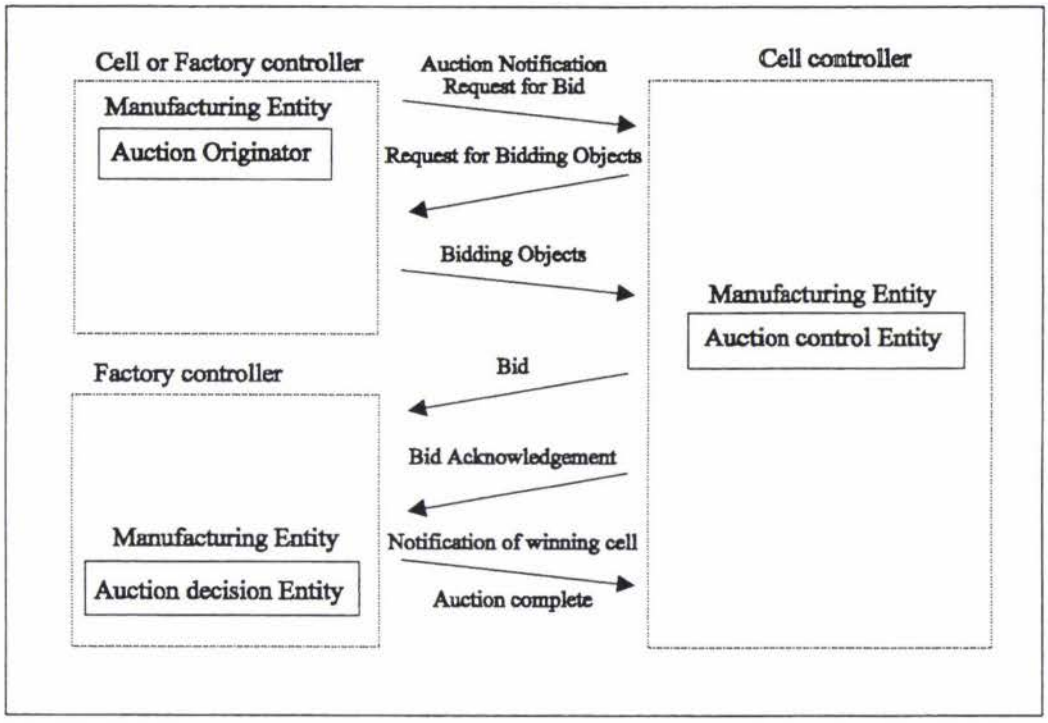



Figure 3.10
Bid messaging process

A simple bidding process was devised to maximise the autonomy of each of the entities involved in the auction. Through devising a system where the message passing provided the coordination between the entities, an asynchronous communication methodology was employed, removing the need for a synchronisation overheads. Figure 3.10 shows the messaging process as it was designed for the FMS control system. As shown in the figure 3.10 three entities are involved, the auction control entity at the cell controller, the auction originator which could reside at the cell control layer or at the factory controller and the auction decision entity, this entity at the factory controller.

### 3.7.5. Auction Co-ordination

In the case of the FMS control system the initiation of the auction or the "Bid Request" can originate from either the cell control layer or the factory controller. The factory controller can initiate an auction if a new task is to be released into the system. A cell controller can initiate the auction process if it has a task which requires processing. Therefore in a FMS control system several entities could be wanting to undertake an auction at any one time. Thus the design of the auction based system had to take this into account. It was assumed that only one auction of any one type could take place at any one time. Therefore a method or coordinating the auctions at the cell control level was required, this co-ordination method would require only one auction to take place at once. For this co-ordination a token passing method was utilised, here a token is passed around all the entities in the factory which may be required to initiate an auction, as shown in figure 3.11. If an entity has any tasks that require auctioning, then it will take the token for the duration of the auction.

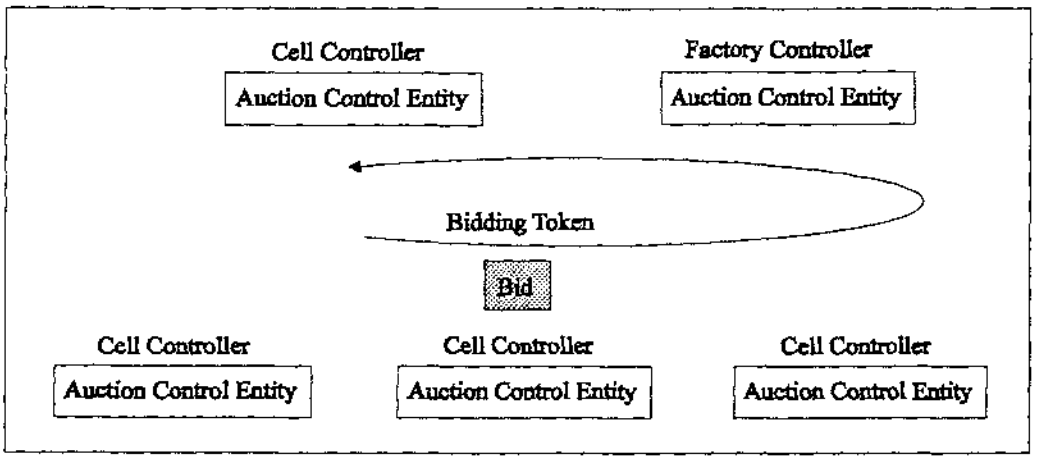



Figure 3.11
Bid Token passing around factory

The method of using a bid token to co-ordinate the auction process also has other advantages, the bidding token can be transferred around the factory as required. The bid token can be allocated on a priority basis, ie entities that have a larger number of tasks to bid in their queues can be serviced by the bid token more often. A number of rules were formulated in order to help maintain the desirable characteristics of the hybrid control model.

1. Only one auction can take place at any one time on the network.

2. Every entity wishing to take place in the auction must conform to the messaging protocols which define the passing of the token amongst entities.

3. Each entity can only hold onto the token for the length of one auction, or a maximum process time. If that maximum process time is violated then it will be assumed that the entity has failed and the token will be passed on to the next entity.

4. An entity can only initiate an auction if it has possession of the bid token.

### 3.7.6. Bid Formulation

The actual formulation of the bid to which all the bidding entities will be evaluated upon is a critical calculation. As mentioned previously the effectiveness of the bidding system is directly related to the quality of the information to which the bid calculated. To calculate the bid for each cell the relationship or coupling between the cell control and the IE must be relatively tight, thus enabling a reasonable amount of useful information to be transferred in an effective time frame (Shaw 1994). This has been achieved in the model of the previously discussed IE. A direct relationship is drawn between the state of the VMD which resides in the cell controller and the actual state of the low level device.

The decision on which "scheduling" method to employ in the bidding process will affect what sort of information will be required by the entity which is to formulate the bid. Shaw (1987) suggests that using earliest finish time as a basis for the formulation of the bids in an auction based system, provided the best performance in terms of late jobs, in process wait time, tardiness and mean flow time.

Thus the calculation of a bid for each cell would be based upon earliest finishing time. This required a variety of information about the number of jobs left to be processed, the set up time of the job, the number of parts to be processed and so on. A basic model, Figure 3.12, was then formulated to describe the calculation of this bid based upon the information available.

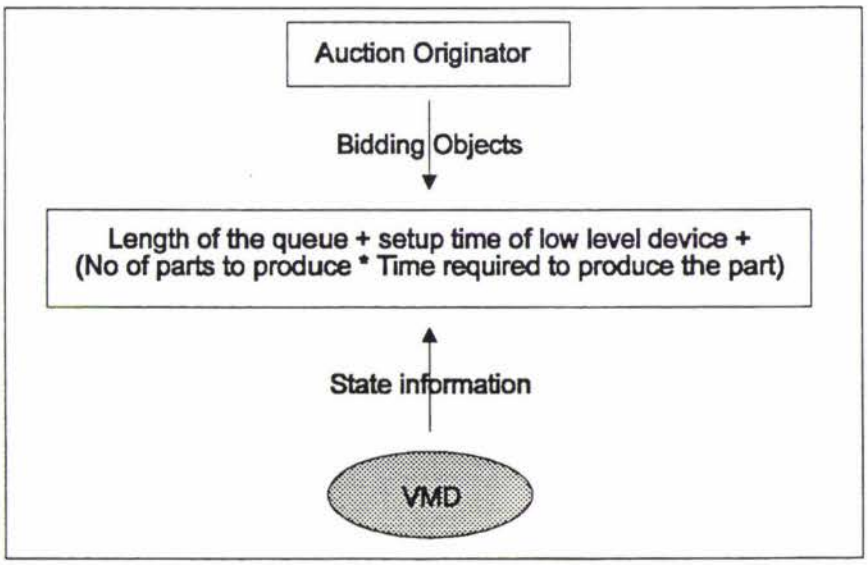



Figure 3.12
Bid calculation model

## 3.8. Conclusion

In this chapter the formulation and design of a model for which to base the implementation of the Flexible Manufacturing Control System upon has been discussed. The control model consists of three separate layers and is a blend of both heterarchical and hierarchical control architectures. The top most layer consists of a single controller, the factory controller provides resources and goals for the next layer down, the cell control layer. This layer consists of a number of controllers linked together via a LAN. Each controller is responsible for directing a number of IE's under its jurisdiction. Jobs within this model are distributed through the use of a opportunistic scheduling methodology. This system uses a negotiation process to determine the distribution of jobs amongst the cells within the factory.

# CHAPTER 4

## REQUIREMENTS AND TOOLS FOR THE IMPLEMENTATION OF THE FMS CONTROL MODEL

Table of Contents

## 4.1. Introduction

Previous chapters of this thesis have discussed the background and theoretical models upon which the FMS control system is based. This chapter will take a step closer to the physical system and will discuss the tools that were selected for the implementation of the Flexible Manufacturing Control System. The evaluation of prospective tools capable of aiding in the implementation of the control system, was undertaken with the consideration that the control model itself consisted of three distinct layers. The actual evaluation of these tools was conducted using a series of criteria drawn from the major goals of the research. A number of tools were selected, based upon how they were able to satisfy the implementation requirements.

## 4.2. Implementation Requirements of the FMS Control Model

Discussed in chapter 3 were operational considerations and characteristics to which various portions of the FMS control model must adhere to, so as to maintain the objectives outlined for this research. However two important considerations not yet discussed were drawn from the original aims of the research. These being:

1. Utilise the existing FIX DMACS SCADA software package as extensively as possible.

2. Time limitations, it was important in the early stages of the research project to demonstrate results so as to ensure the future of the project.

Time limitations, required that the research being undertaken relatively quickly to demonstrate worthwhile results, at an early stage. As it was recognised that the scope defined for this thesis was quite ambitious considering the time frame. This required several compromises between what would possibly be the best option in the long term and what would give the best results in the shortest length of time.

In order to satisfy these implementation requirements yet maintain the integrity of the design phase through to the implementation, three tools were selected. The first of these tools was FIX DMACS, one of the major aims of this research was to determine the extent to which FIX DMACS could be utilised in the implementation of a FMS control system. It was however recognised early in the design phase that FIX DMACS lacked the flexibility to undertake all phases of the software implementation of the FMS controller. Thus a programmable language called Visual Basic was also selected which was capable of creating custom applications that could interface with FIX DMACS within the same environment. Finally a programmable device was required for which to implement the IE within. The concept of the IE is such that the design can be implemented within a variety of different programmable devices, one such suitable device had to selected.

# 4.3. FIX DMACS

## 4.3.1. Introduction to the FIX DMACS Software

In the initial stages of the project a software package call FIX DMACS was purchased within the Department of Production Technology and was made available to the project researchers. The FIX-DMACS (Fully Integrated Control System-Distributed Manufacturing Automation and Control Software) software was a Personal Computer (PC) based SCADA (Supervisory Control and Data Acquisition) package with a very powerful user interface and integration capabilities to other PC based software packages.

FIX DMACS is a software system capable of running on an IBM PC. The core software of the system operates to allow specific FIX DMACS applications (Figure 4.1) to perform the tasks assigned to them. The most basic of these tasks is data acquisition and data management. "Data Acquisition" (Intellution Manual, Software Guide) is the ability to retrieve data from the plant floor and to process that data into a useable form. FIX DMACS can also write data to the factory floor, thus creating a two way communications link. FIX DMACS communicates directly with the hardware it is controlling, such as a Programmable Logic Controller (PLC). Once FIX DMACS acquires data, it manipulates and channels the data accordingly to the requests of the software applications or tasks, this process is known as "data management".

It operates in a similar fashion to other SCADA packages comprising of a central database that defines how, when, where data is collected. FIX DMACS also has various functions referred to as applications that use and manipulate the input and output data to and from the database.

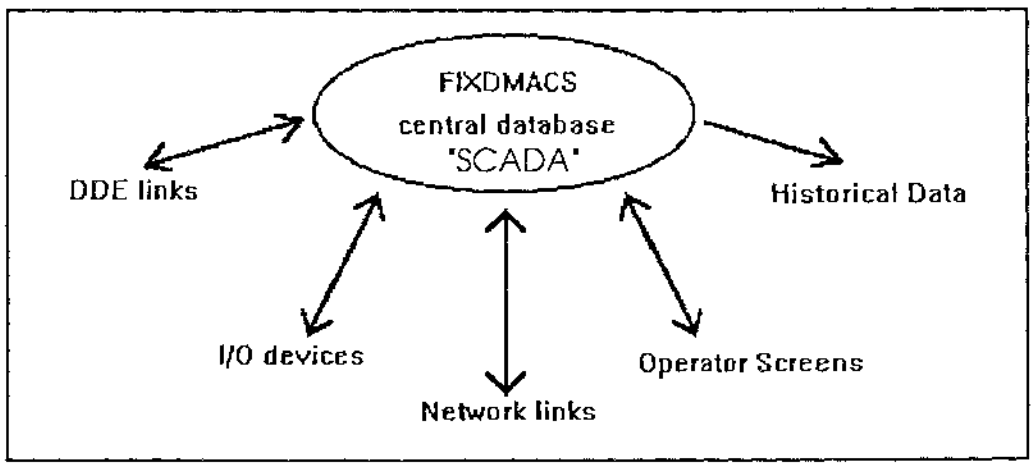



Figure 4.1
Diagram of FIX DMACS applications

#### 4.3.1.1. DDE Links

DDE links provide the continuos transfer of real time data from FIX DMACS database blocks to other applications that support dynamic links. DDE links are only available to applications operating within a Micro soft Window's environment on an IBM compatible PC.

#### 4.3.1.2. Historical Data

FIX DMACS provides a facility for the collection and presentation of historical data for future analysis. Historical data is important for trending and displaying data for long term decision support.

#### 4.3.1.3. I/O Devices

This is the channel by which FIX DMACS communicates to the physical manufacturing environment. The actual link involves the definition of an I/O driver that communicates to a programmable device and thus to the actual physical machinery itself connected to the machinery.

#### 4.3.1.4. Operator Screens

A Draw application, can be used to create operator view screens and a View application views these screens created in Draw. These view screens can be utilised to display data collected from the factory floor. This enables a representation of the portion of the factory currently being controlled, to be drawn, then dynamic links can be added to emulate the plants physical movements on the operator screen.

#### 4.3.1.5. The Database

The database function in FIX DMACS consists of functional database blocks eg figure 4.2. Each database block performs a particular task and thus every database block (with the exception of a few) requires scan resources and hence processing time.

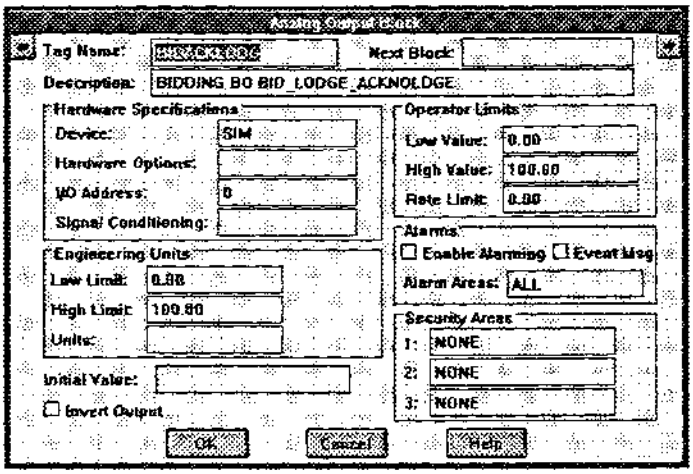



Figure 4.2
Example of a FIX DMACS database block

### 4.3.1.6. SCADA Functions

The SCADA functions of FIX DMACS are central to the operation of the software as FIX DMACS is basically a SCADA package. What a SCADA package does is to replace the original manual control room with a virtual control room based on a PC. This means all the original functions that used to be done manually can now be done by computer software and thus can be more easily controlled. The main SCADA functions of FIX DMACS include:

- Monitoring
- Supervisory control
- Alarming
- Control
- Reporting
- Multi application and distributed communication

Monitoring is the ability to display real-time plant-floor data to operators. Supervisory control is the monitoring of real-time data coupled with the ability of the operator to change set points and other key values directly from the computer. Alarming concerns the ability of the software package to recognise in real-time exceptional events within the process and to report these events to the operator. As FIX DMACS can operate in a distributed nature alarms can be generated and transported over the LAN to physically separate points. Control is the ability to automatically apply algorithms and adjust process values and thereby maintain a system within set limits. This control goes one step further than supervisory control, in some cases it removes the need for human interaction, through using the computer to control the entire process.

The reporting functions are higher information services that can archive or store real time data for analysis, the analysis can either be done using the reporting functions of

FIX DMACS or can be transported to another application via industry standard exchange protocols.

#### 4.3.1.7. Distributed Processing Capabilities

FIX DMACS has the capability to operate on either a PC as a stand alone node or to operate on a network or LAN as a series of nodes ie Figure 4.3. When operating as a series of nodes on a network, data can be transported around the network, thus allowing the system to become distributed. In this fashion several SCADA nodes can control portions of a factory at physically separate locations yet information services can be organised to gather data for reporting purposes at any one individual node.

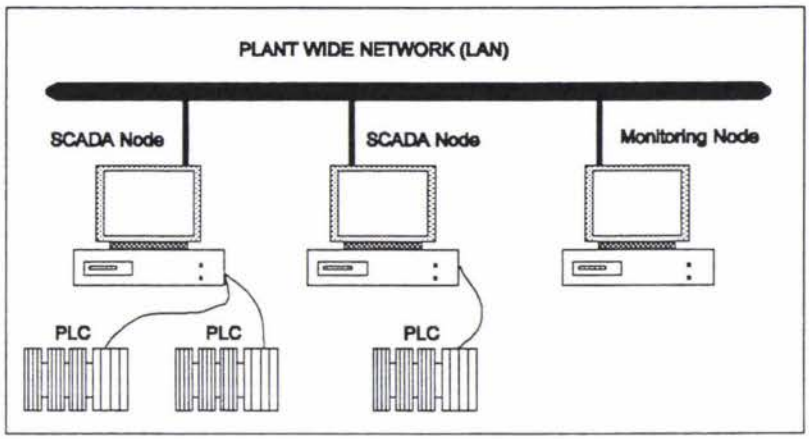



Figure 4.3
Distributed layout of FIX DMACS system

Further information about the capabilities of FIX DMACS is available in (appendix A.1).

### 4.3.2. Evaluation of FIX DMACS

Even though FIX DMACS was central to the implementation phase of the research it was still important to evaluate its capabilities in terms of the implementation requirements and the design so as to determine the extent of its usefulness.

To evaluate FIX DMACS a series of evaluation criteria were established. These criteria were chosen based firstly on the earlier discussion of the desired characteristics mentioned during the formulation of the control model for the FMS control system. Secondly the criteria were based upon the requirements expressed in section 4.2. The criteria were as follows:

1. The ability to implement as much of the FMS control model as possible.

2. To be able to operate in a distributed fashion.

3. To provide a friendly and graphical interface for the user and the programmer.

4. Capable of being integrated along with other applications into a single system with minimum effort.

5. The ability to transfer information and communicate with other applications.

Once the above criteria had been formulated the next step was to use them evaluate FIX DMACS.

#### 4.3.2.1. Evaluating FIX DMACS Control Ability

The initial investigation into FIX DMACS revealed many promising features especially for application in the middle to lower regions of the control model. Because FIX DMACS was designed as a SCADA package it had the ability to interface to a wide variety of different low level control devices, such as PLCs. This interfacing with low level devices was imperative to the implementation of the FMS control model. Secondly the ability of FIX DMACS to interface with a large variety of different low level devices fell into line with the project's objectives to design a generic controller able to control a wide variety of low level devices.

FIX DMACS also had the ability to collect, store and manipulate a wide variety of data from the low level devices, this provides the ability of the controller to "see" what is happening at the low level device and if necessary construct a "virtual model" of the current state of the low level device. Thus enabling an effective implementation of a VMD.

FIX DMACS has the ability to display collected data using one of the applications which FIX DMACS is composed of. This enables a wide variety of information to be displayed in a number of different formats. Such as bar and line graphs, SPC charts, cumulative graphs and diagrammatic representations of real factory floor devices. However, even though the information processing and display capabilities of FIX DMACS were quite powerful, it was designed mainly for comparatively simple display tasks. Therefore it was decided that FIX DMACS could potentially be used to implement many of the simpler information presentation tasks and operator screens required by the FMS control system.

The information collected by FIX DMACS, not only could be used to inform users about the status of the process in question, it can also be modified and used to control the process itself. By arranging units of the "database" in certain ways, control decisions can be made based upon the value of the information being collected. This gives FIX DMACS the ability to alter the process according to what the designer and operators intend. These control aspects of FIX DMACS venture beyond however the usual PID or step change controllers, certain database blocks are available that can be programmed to manipulate data based upon the flow of the program. This gave FIX DMACS the ability to provide the supervisory control ability required both at the cell and factory control layer.

#### 4.3.2.2. Evaluation of Distributed Operating Characteristic

Due to the hybrid nature of the FMS control model, any tool that was to be used as the basis for which the control model was to be implemented within must have the ability to operate in a distributed nature. For example the cell control layer does not in fact consist of a single cell controller but many physically distribute cell controllers, all of similar construction and all able to co-ordinate the operation of the low level device controllers below it and still be able to communicate over a LAN. The "open architecture" of FIX DMACS allows individual nodes to not only communicate over a network to other nodes on the network, it also provides the ability for an application external to FIX DMACS (eg Micro soft EXCEL) to gain access to information on nodes, other than the one which the application to operating upon. This enables a series of cell controllers to operate individually on the cell control task at hand, yet still be able to negotiate or communicate via the network amongst themselves or to the factory controller (a FIX DMACS node).

#### 4.3.2.3. Evaluation of User Friendliness

One major strength of FIX DMACS compared to other similar SCADA available, was the user friendliness of the program. Even though FIX DMACS is a complicated tool, it has been developed within the Window's environment and thus has the advantages of a well-conceived graphical user interface. Windows is currently the fastest growing and most favoured environment for most users of the IBM compatible computer. This coupled with well written documentation and excellent product support in New Zealand, meant that most problems when encountered could be solved quickly and easily. Further more the time required for a user to become familiar with the package and thus be able to make use of it appeared to be relatively short.

#### 4.3.2.4. Evaluation of Inter-application Integration Capability

FIX DMACS is a SCADA application that is capable of operating in the Micro soft Window's environment. As previously discussed this environment was designed in order to provide a user friendly graphical interface. Another great advantage is that all applications that operate in this environment maintain a similar interface, ie all. Thus rather than having several applications operating many different user interfaces, all the applications used in the FMS control system will be written to operate in the Window's environment and thus will have a consistent interface.

#### 4.3.2.5. Evaluation of Inter-application Communication Capability

The communication capabilities of FIX DMACS were very important in enabling it to play a part in the implementation of the FMS control system. The control system in many respects is simply a system of loosely coupled applications that operate individually but towards a common goal. For this reason information transfer between the various applications within the control system is very important. FIX DMACS has three major channels for information to flow in both directions each channel having its own merits.

One method that can be utilised to transfer information not only between physically separate FIX DMACS nodes but also to other applications either on local or remote nodes can be achieved through the use of Dynamic Data Exchange (DDE). DDE is a protocol for transferring information between different applications within a Window's environment. The DDE method works by specifying a link between two applications within the Window's environment, either of the two applications can receive or transmit data.

A similar communications method to the one explained above is available called Easy Database Access (EDA). EDA consists of a series of function calls, available to a user defined program that will allow the programmer to manipulate information within a database. This method is similar to the previously one in that it also can operate over the network, although it operates much faster and can handle larger amounts of information than DDE.

The final path through which information can be channelled to and from FIX DMACS is through using disk based files as an intermediary storage point for information. Information can be written to a file by one application and then accessed by another application at a later date.

### 4.3.3. Conclusion of FIX DMACS Evaluation

At the conclusion of the evaluation It was felt that overall FIX DMACS had the ability to undertake several functions or tasks required in the implementation of the FMS control model. The functions that it is capable of undertaking would other-wise take considerable effort using traditional programming methods. FIX DMACS does however have several advantages when compared to similar applications of an equitable cost. These being; high applicability to the implementation of the FMS controller, ability to operate in a distributed nature, communication abilities, excellent user interface and control elements that can be programmed.

However, despite these advantages, FIX DMACS does have several major disadvantages that limit its applicability to the FMS control model. FIX DMACS is inherently a process supervisory control package and does not allow the flexibility required when using it to control a discrete manufacturing environment. For this reason the programming elements available to build custom control blocks are very limited and cumbersome to use. Finally graphical interfaces that can be constructed using the Draw program within FIX DMACS lack the sophisticated functions required to display some of the information required to give a fair representation of the state of a manufacturing environment.

FIX DMACS was only suitable for the implementation of some portions of the FMS control model. It did not have the flexibility nor controllability to provide all of the factory controller's functions, nor was it capable of displaying some of the factory information in an expectable format. As FIX DMACS is a supervisory control package it obviously does not have the ability to provide the low level, real time control necessary for the control of a shop floor device. For these reasons several other tools were required to aid in the implementation of the control model.

## 4.4. Programming Language

### 4.4.1. Implementation Requirements of the Programming Language

FIX DMACS was found only capable of providing the ability to implement parts of the factory and cell controllers as well as the Intelligent Entity. For this reason it was decided that a number of other software applications would be required to fully implement the FMS controller. These off-the-shelf applications would include CAD and CAM software. However, to provide a single user interface to these applications and to fully implement the functions required by the factory and cell controllers not supplied by FIX DMACS, it was recognised that a software programming language would be required.

In order to select a suitable programming language a number of requirements had to be fulfilled. These requirements are based upon the original aims of the research coupled with the characteristics of the FMS control system and portions of the control model that are impossible to implement using FIX DMACS. These requirements were;

1. To be capable of transferring large amounts of information within the network, to and from both factory and cell controllers.

2. The programming language must be able to provide a user interface that utilises Micro soft Windows as an environment. This is to both provide consistency between the many applications that the FMS control system will consist of and also to create a user friendly environment.

3. The programming language used for the implementation must not significantly impact upon the overall cost of the project, keeping within the aim of providing a low cost control system.

4. Be able to communicate directly to FIX DMACS and other applications within the Window's environment, thus being able to integrate and communicate between different applications.

5. The software programming language should be as flexible as possible and be capable of early being altered as the need may arise.

6. The ease of use and ability to provide results in the shortest possible time.

Once these requirements had been formulated it was then possible to evaluate several potential programming languages and select the best one. The programming languages that were evaluated include; Micro soft C, Micro soft Visual C++, Borland C & C++, Micro soft Visual C, Micro soft Visual Basic. The major points of consideration were as mentioned previously with special emphasis placed upon the ability of the programming language to provide an easy to use Window's Graphics interface. The evaluation resulted in the selection of Micro soft Visual Basic (VB) as

the programming language being able to satisfy the greatest number of the previously mentioned requirements.

### 4.4.2. Introduction to Visual Basic

Visual Basic is a programming language written by Micro soft that operates in a Window's environment and thus has all the benefits of a Graphical User Interface (GUI). The origins of Visual Basic can be found with a basic programming language supplied with MS-Dos called GW-Basic. The core of the Visual Basic language has existed for several years and is an improved version of GW-Basic called Quick Basic. However, Visual Basic has improved on previous versions of Basic and now can be considered a 'structured' programming language much like other languages such as C or Pascal. Visual Basic is a programming environment in which you define not only the 'program' or operation of the application but you also define the interface of the application. Due to Visual Basic being tightly coupled to the structure of the Window's environment it is possible to create applications which integrate well into the with other Window's applications.

Visual Basic is an event driven programming language, this means that rather than traditional programs which operate sequentially. Program code written in VB operates in response to an "event". An event is most commonly caused by the interaction of a user with the GUI. This results is a very different programming style than traditional programming methods, in VB the user interface is first defined using a tool box supplied with VB (figure 4.4), then the code is added.

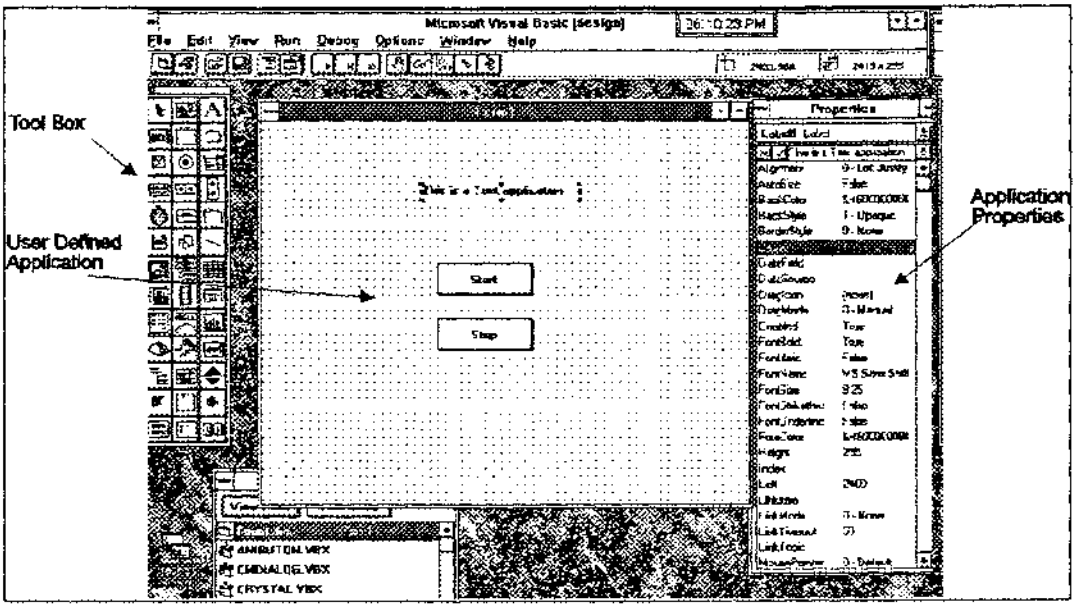



Figure 4.4
Screen dump of VB with a test application

The creation of the GUI in Visual Basic (figure 4.4) consists of utilising the tool box to arrange a number of objects inside a window then setting the properties for each of the objects. If required programming code can then be added to an event that may occur due to the user's interaction with the objects in the window, eg pressing a button will cause an event, this could run a small program that closes the window. A number of windows can then be joined together in certain ways to create an application. Programming in this way with VB is a very simple and powerful way to create Window's based applications in a short period of time.

## 4.4.3. Evaluation of Visual Basic

Visual Basic was evaluated along with a number of other programming languages mentioned earlier. As a result of this evaluation it was decided at this stage in the research that Visual Basic would be the best tool to use to aid in the implementation of the FMS control system. The investigation was based upon the requirements outlined previously and the following evaluation of these requirements was undertaken.

### 4.4.3.1. Evaluation of Communication Capability

Visual Basic was capable of using both DDE and the Easy Database Access as well as files as methods of transferring relatively large amounts of information both between applications and FIX DMACS. Using the EDA appeared the best method of transferring information to and from FIX DMACS, unfortunately the EDA for VB was not available for several months into the project and thus the DDE method was initially used. It was possible to use the FIX DMACS network application to

communicate on the network via EDA or DDE from a VB application, without having to program an Ethernet network interface for the application.

#### 4.4.3.2. Evaluation of User Interface

As mentioned previously VB is a Window's based programming environment and thus it was possible to create an interface in a VB application that was not too different from the interface used in FIX DMACS and other Window's applications. This was important as a VB application had to be written which would provide the front end and the "glue" to stick the numerous Window's applications that made up the FMS control system together. This enabled a consistent interface to be created for the FMS control system. The GUI created by VB was also easer for most users to operate as most had used a Window's environment before and were thus familiar with its operation.

#### 4.4.3.3. Evaluation of Package Cost

The cost of Visual Basic Professional version 3 was approximately $750, this provided all the application tools required to successfully produce stand alone executable Visual Basic applications which operated in the Window's environment. Once applications had been created in VB no organisation except Massey University had any rights to the software the cost was a one off and would not need to be repeated at a later date. Thus maintaining the aim of a low cost solution to the FMS control system.

#### 4.4.3.4. Evaluation of Programming Flexibility and Ease of Use

VB is a relatively flexible programming tool and is capable of producing all but the most complicated applications. It does have several limitations however in this area, both in speed of operation and flexibility. The creators of VB were successful in producing an easy to user programming tool for Windows, however sacrifices were made along the way which limit the power and flexibility of the tool compared to the other programming languages investigated. It was felt however despite these limitations that the flexibility and speed of operation was sufficient for the task required.

Reiterating on the easy of implementation and easy of use is important as these were major points in the decision to use VB for the implementation of the FMS control system. Due to the easy of use, VB applications can be created in a shorter time frame than any other similar tool. This meant that results could be shown early on, thus maintaining interest within the project.

After the investigation had concluded it was realised that even though VB would most definitely fulfil the short term aims of the project, limitations in flexibility and speed of operation may prove limiting factors later on. Thus it was decided that VB would be best used as a prototyping tool as it is often used in the software industry. This would enable a working prototype to be produced in a relatively short period of time.

### 4.4.4. Conclusion of Evaluation of Visual Basic

The evaluation of a number of programming languages was conducted in order to determine the suitability of the language in question. It was determined that in the short term using Visual Basic would be the best option. VB offered quick professional results together with the ability of utilise several different channels for communicating between applications. This was able to be done within a Window's environment, which enabled a common interface to be used for all components of the FMS control system. Furthermore this was all possible without many of the usual disadvantages of a Basic programming language.

## 4.5. Low Level Device Controller

### 4.5.1. Implementation Requirements for the Low Level Device Controller

The low level device is a physical machine that resides at the machine level and undertakes operations on physical objects, a low level device as its name suggests has little or no "intelligence" ie little computing or control ability. A controller is necessary to regulate the actions of the low level device, because of this a PLC or such similar controller is necessary. This control device will be required to be capable of implementing the 'Intelligent' layer of the IE.

One of the major aims of the research was to provide a tool for a manufacturing environment that was flexible in terms of it operational ability as well as its applicability to a wide variety of situations. For this reason it was important to design such a tool that would be able to utilise a variety of different low level device controllers. This meant that the FMS control system would not be tied to one particular type of low level device controller. Therefore is one capability which FIX DMACS was able to bring to the implementation of the FMS control system. Thus when deciding upon what type of low level device to use, it was not as important to decide upon a single type that would be used exclusively from then on, as it was to find a capable device that was easy to use and did not cost too much. Thus of the three brands of PLC like devices evaluated any one of them could and in fact was used for the implementation of the low level device, the choice of the type of controller needs to be determined for each individual application.

Requirements were formed in order to evaluate the applicability of potential low level device controllers to the implementation of the FMS control model.

1. The cost of the low level controller is required to be kept at a minimum.

2. The controller must have the capability to implement the required tasks of the lowest level of control according to the FMS control model.

3. The controller must be easy to program and to modify.

4. The brand and type of controller should be well known and technical support should be available.

Using the above criteria several PLC like devices were evaluated. Unfortunately due to financial restrictions the number of potential devices was somewhat restricted. The devices investigated were; Allan Bradley PLC's a MILSIC A Mitsabishi PLC and a Fisher & Paykel Programmable State Controller (PSC). The Mitsabishi PLC was found to not be capable of undertaking all of the tasks required, it had a limited memory for storing data and had only a limited number of Input and Output I/O modules. Furthermore its programming interface was quite difficult to use and was tremendously slow.

The Allan Bradley PLCs had good programming interfaces to the PLC modules however the actual programming environment was still difficult to use at times. The capabilities of the PLC's in most cases far exceeded what would be required for the implementation of parts of the FMS control system. Unfortunately the cost of these devices was quite significant in comparison to the other hardware used in the research and therefore the Allan Bradley PLCs became second choice behind the Fisher & Paykel PSC.

## 4.5.2. Introduction to Fisher & Paykel PSC

The Fisher & Paykel PSC is a PLC device which instead of ladder logic it uses a State Programming Language (SPL). The SPL is hailed by Fisher & Paykel to be a much easer to understand programming language when compared to traditional PLC languages. State logic operates in a similar fashion to a flow diagram. Only one state can be active at any one time, when the current state is the focus of the program it runs the code relevant to that state then moves onto the next state, eg figure 4.5.

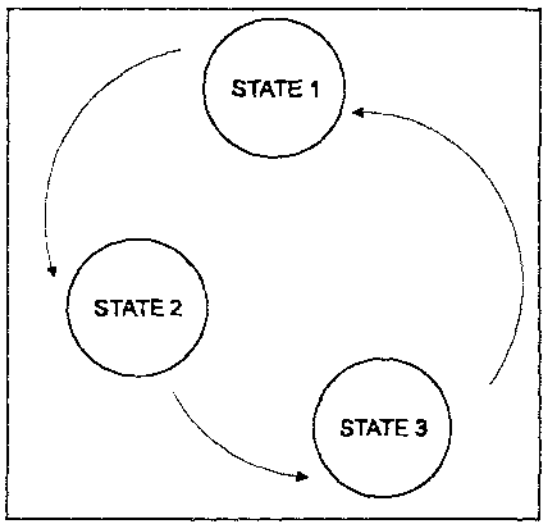



Figure 4.5
Example of state logic program flow

The SPL it is easer to implement the logic within PSC than a standard PLC using ladder logic. This decreases development time and enables maintenance to be undertake with greater ease. Programs written in state logic are also easer to debug and document. The Fisher & Paykel PSC itself is a reasonably capable control device able to operate 64 separate control loops simultaneously, at any one time. One control loop consists of a number of states, of which only one state can be operational at any one time. The order in which the loop moves from state to state depends upon the logic of the loop and in the particular way in which the loop has been constructed.

Due to the PSCs speed of operation it is very suitable for the real time control, of time critical operations, in fact it would be possible to use the PSC to control several low level devices at any one time. The speed of operation of the PSC is relatively quick, with a typical scan time of approximately 3.5 ms. The PSC also has up to four RS-

232 ports, one of which must be used to communicate with the cell controller, the others however can be used to communicate with the Low Level devices via a RS-232 connection if available. The I/O units available for the PSC are many and varied. There are around 15 different types of I/O modules ranging from simple 24 volt relay switches to 12 bit 4 channel digital to analog converters. More detailed information on the PSC is available in the Fisher & Paykel Hardware Manual.

### 4.5.3. Evaluation of Fisher & Paykel PSC

The evaluation of the Fisher & Paykel PSC was conducted in accordance with the criteria mentioned in section 4.5.1 that discussed the requirements for a low level device controller.

1. The cost of the Fisher & Paykel PSC in comparison to the other controllers was relatively favourable. The cost of the processor module with 64k or memory a LAN card, power supply, 3 input modules of various types and 4 output modules were directly comparable to that of the other systems evaluated.

2. The capabilities of the PSC were evaluated in terms of the ability of implementing the relevant portions of the IE. To do this the speed, power, size of useable program, storage for factory data, size of instruction set and the programming instructions were all investigated along with the capabilities of the available I/O modules. The result of this investigation determined that the PSC was indeed capable of the implementation of almost all of the tasks required by the Low level device controller.

3. & 4. Whilst the programming of the PSC was not in any way easy relative to higher level languages such as Visual Basic or C++, it was however easier to use than the similar programming environments associated with the other PLCs. This was due not only to the fundamentally simpler programming method used (Programmable State Language) but also the application that was used as an environment to program the PSC was relatively easy to use.

### 4.5.4. Conclusion of PSC Evaluation

Although any of the programmable devices evaluated for the implementation of the IE could have been used, it was felt that the Fisher & Paykel PSC was the best option. The PSC provided an easy to use programming environment, which reduced development time. It was also cost competitive to the other options explored. Its performance was comparable to the other programmable devices and there was excellent product support available through Fisher & Paykel.

## 4.6. Conclusion

In this chapter the major tools required for the implementation of the flexible manufacturing control system model have been discussed and evaluated. This evaluation was done with consideration given to the original aims of the research. These aims include minimising the cost of the implementation of the control system, minimising the time required to undertake this implementation, attempting to produce a user friendly graphical interface to the control system and to maximise the flexibility of the control system. For this purpose three major tools were selected to implement parts of each layer of the three layer model described in section 3.2.3.

The first of these tools is a SCADA software package called FIX DMACS, this software package compared to other similar packages provides a simple easy to use and program interface, which provides for wide applicability to the control model. This software package also allows the flexibility and customisation required for the implementation of the FMS control model. FIX DMACS is capable of implementing most tasks required of the cell control level and some of the tasks required of the factory. Limitations of FIX DMACS due to its restricted program ability, inability to transfer large amounts of information and its inflexibility meant a second tool was required which would be able to fill in the gaps left by FIX DMACS in the cell and factory control levels.

Micro soft Visual Basic (VB) was chosen for this task. VB is a basic programming language that enables a programmer to relatively easily create applications within a Micro soft Window's environment. This had numerous advantages as Windows was the environment that both FIX DMACS and any other applications required for the control system would be operating within. Thus a user friendly and consistent interface was able to be created.

The final level of control that was required to be implemented was the lowest level of control, the implementation of the Intelligent Entity. One of the original aims of the research was to create a control system as flexible as possible, for this purpose it was decided that the implementation of both the cell control and IE should be independent of the type of low level controller used. This was possible through creating a generic interface between the IE and the cell controller. A low level device controller was chosen which could be obtained for minimum cost and was able to undertake all the tasks required of it. This however did not effect the long term direction of the low level controllers implementation.

# CHAPTER 5

## THE IMPLEMENTATION OF THE FLEXIBLE MANUFACTURING CONTROL SYSTEM MODEL

Table of Contents

# 5.1. Introduction

This chapter discuses in full, the resulting implementation of the three layered Flexible Manufacturing System Control Model. The physical model is the basis for the implementation of the Flexible Manufacturing control system, as it represents the physical operation and structure of the system. The physical model itself was formulated and discussed previously in section 3.2.3. "Proposed physical model of a hybrid FMS controller". As of yet not all of the FMS control system has been implemented, thus one aim of this chapter is to document the current state of this project. Each individual entity that has so far been implemented is described and its operation explained.

During the implementation portion of this research the major aim of the development was to try and implement as much of the FMS control system as possible so as to produce a skeleton system that would if possible prove the concept of a hybrid architecture. Thus rather than attempting to fully implement any one layer of the control model, parts of all three layers have been implemented.

The implementation of the three layered model will be discussed by separating the system up into three separate parts, the Factory controller, Cell controller and Intelligent Entity. These three separate components will then be investigated separately at first to explain the function of the components, how they were implemented and secondly to explain how the components react with each other.

## 5.2. The Implementation of the Three Layered Flexible Manufacturing Systems Model

### 5.2.1. Flexible Manufacturing System Control Model Assumptions

A number of assumptions were made during the formulation of the design and physical model implementation, to allow the actual system to be implemented in accordance with the model. Further more a number of key terms were required to be defined in order to ensure consistency and to define the boundaries of the system. These assumptions and definitions are listed as follows:

Definitions

1. A **Manufacturing Entity** is "what ever is required to accomplish one or more manufacturing activities" (Weston et al 1991). Manufacturing entities are the building blocks of a manufacturing system, a FMS control system would consist of a number of MEs. Although there are wide functional differences between MEs, all process information and thus will ultimately make software based state changes.

2. A **Job** consists of a series of tasks, each task will be performed in the order in which it appears in the job list file.

3. A **Task** is an operation performed on a work piece by an individual low level device.

4. The **job list file** or **IFFile**, is a sequential list of all the tasks within a job. The order in which the tasks appear in the job list file is determined by the operator who defines the job.

5. The operations that the low level device will perform as required by a particular task can be adequately described through a IFFile that contains:

   - Fixtures information
   - Tooling information
   - Machine control information
   - Task description information
   - Next task rebidding information

6. A **Low Level Device** is a physical device that exists upon the factory floor such as a lathe or mill and performs an operation upon a work piece. This operation alters the physical state of the work piece to a more desirable state.

7. An **Intelligent Entity** consists of a programmable device or low level device controller and a low level device. The IE is responsible for implementing the Virtual Manufacturing Device's (VMD) services at the low level device.

8. The **VMD** is an abstract representation of the common characteristics of the low level manufacturing devices and represents externally the behaviour and state of the device. This representation also includes services that enable the low level device to be effectively controlled.

9. A **Cell** is a collection of IE's. No two IE's within the cell will contain the same low level device. There will be a maximum of four IE's in any one cell.

10. The function of the **Cell Controller** is to co-ordinate the MEs under its control. This entails monitoring, supervisory control and scheduling for the IE's within its boundary. A cell controller is capable of simultaneously processing up to four different tasks at any one time whilst maintaining the other necessary operations of the cell controller.

11. The **Factory Controller** is the controller responsible for the master scheduling and monitoring of the cells within the factory.

12. An **Auction** is the process of Scheduling an individual job in an opportunistic fashion. An Auction is originated by either the factory controller a cell controller and all capable cell controllers place Bids with the factory controller for the current task. The factory controller then allocates the task to the cell according to a decision criterion.

13. **Bids** are the offers or "prices" placed by the cells to the auction originator, after receiving information about the object of the auction.

14. A **Bidding Object** is a piece of information relating to the object of the auction. Several bidding objects are required to allow a cell to gain sufficient information about the task that is the object of the auction to place a bid.

15. The **Auction Originator** is the ME which called for an auction to take place, it is the ME which broadcast the Bid Request.

16. **Bid Request** is the data object which informs other MEs that an auction is taking place.

## Assumptions

1. All resources except tools and fixtures which will be required to process a task will be available when needed.

2. The Tools and Fixtures a low level device may require are stored in the local cell, thus no transportation will be required for these resources.

3. Loading of the Tools and Fixtures at the cell level will occur automatically when requested. Conformation that this task has been completed, is required.

4. Transportation of a part associated with a particular task will occur automatically when requested and in a pre defined period of time.

5. Each cell will contain a maximum of four devices with only one of any type of device present in a cell.

6. The low level device can be controlled adequately with the existing VMD services via the low level device controller (IE).

7. If the machine control file is not in a form which the low level device requiring it can understand, then a suitable conversion utility will be available.

8. The bids placed by a cell capable of undertaking a task which is the object of an auction, will adequately describe the cells ability to undertake that task.

### 5.2.2. Implementation of the Cell Control Layer

The individual entities of the cell controller are loosely coupled yet interact together to operate as a whole. The various entities of the cell controller have been implemented in either one of two ways, by using Visual Basic or FIX DMACS. Most of the major functioning of the cell was implemented within FIX DMACS. However, the transportation of information between cells was unable to be done by FIX DMACS and thus was done by Visual Basic. This meant, in each cell a number of different applications operate together to form the cell controlling functions.

The most fundamental operation of a cell is to complete a task within a job. Each job consists of a number of tasks, with each task relating to a single operation that can be performed upon a work piece by a low level device. Figure 5.1 demonstrates the fundamental operation of a cell, receiving jobs, processing then dispatching jobs. Therefore a cell's function is to use the low level device under its control to undertake a task within a job and then to pass the job on, so the next task can be completed.

Each cell controller consisted of a FIX DMACS database, which had been defined to provide information gathering and control functions, with the exception of a few information transportation functions that were under taken by Visual Basic. The entities that will be discussed are the entities that were shown to constitute the cell control model in section 3.4. "Cell controller". These being the VMD, Device Control Entity, Auction Control, Bidding Information Transportation, Incoming and Outgoing Queues, Monitoring Entity, Data Transportation Entity and the Control Transportation Entity.

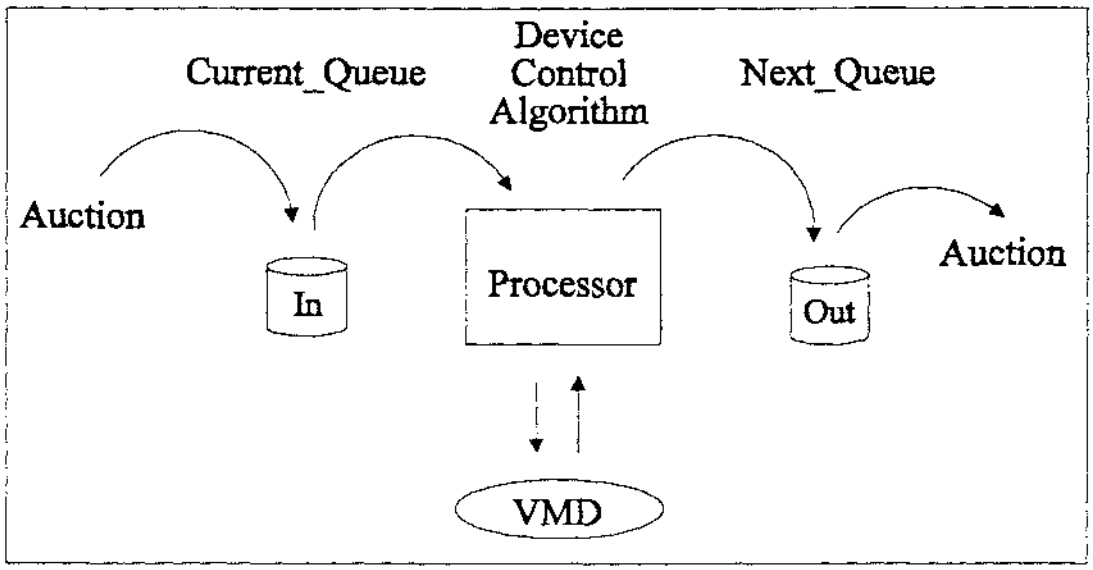



Figure 5.1
Flow of jobs in a cell controller

### 5.2.2.1. Cell Controller Hardware

A cell controller consists of a single FIX DMACS installation, a typical cell controller would be:

- A 486 DX2, IBM compatible PC.

- This computer must be connected to a Local Area Network via an Ethernet card. Further information about the network set up is provided in appendix A.2

- This PC must have Micro soft Windows version 3.11 installed.

- A full installation of FIX DMACS, at least version 3.1.

A more detailed description of the hardware requirements for a cell control is provided in appendix A.3.

### 5.2.2.2. Virtual Manufacturing Device

The Virtual Manufacturing Device is central to the operation of the cell controller. A VMD provides the cell controller with the ability to effect the operation of and to gain current state information of a low level device. The basic concept behind the VMD is to provide a virtual representation in software, of a physical machine. In the case of the FMS control system it was important that the VMD was generic.

The VMD was implemented within FIX DMACS because of its ability to interface with low level device controllers. The building blocks of FIX DMACS were used to combine a number of individual elements that had the capability to reflect the state of a low level device. FIX DMACS database blocks were used for this purpose, each block representing a particular piece of VMD information. These individual blocks were called data objects, as they are objects that represent the state of a low level device. A number of related objects were called a data object group. These blocks could then be linked via a low level control device such as a PLC, to a real low level device such as a CNC lathe.

A core contingent of services would provide the majority of operations required to effectively control any low level device. However, a number of custom services would need to be provided for a small number of operations. It was felt that these services should also be supported by the VMD.

A complete list of all the blocks that the VMD consists of can be found in appendix B. An example of four blocks that represent two objects in a VMD is shown in table 5.1.

Table 5.1
Example of database blocks which make up the VMD

| **Block Name** | **Type** | **Block Description** |
|---|---|---|
| V1MACHBUSY | AI | VMD 1 SO MACHINE_BUSY |
| V1MACHSTOP | AI | VMD 1 SO MACHINE_STOPPED |
| V1PROGREDY | AI | VMD 1 SO PROGRAM_READY |
| V1UNLOADCM | AO | VMD1 SO UNLOAD_COMPLEATE |

These four database blocks undertake four very different operations and conform to a naming convention that was devised to differentiate a large number of database blocks. The Block Name is a unique name given to every database block in the database. The Block name consists of two parts, the data object group type and the function of the block. Further information about database blocks and the naming convention used can be found in appendix A.4.

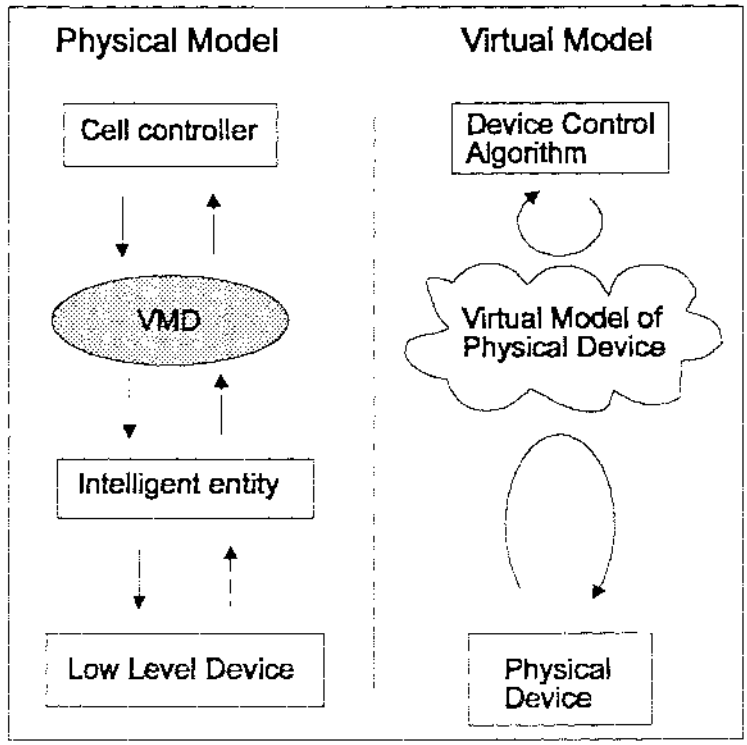



Figure 5.2
Virtual and Physical models of VMD operations

Shown in figure 5.2 is a representation of how the virtual nature of the VMD is utilised. The virtual model provides the link between the device control algorithm and the cell controller. The cell controller can control up to four different low level devices. Each cell controller will contain four VMDs one for each of the low level devices. The contents of the VMD can be altered through three different channels. The first is by an operator configuring the structure of the factory.

The second way in which the information within a VMD could be altered, is through the VMD updating its current state to reflect changes in the state of the low level device. This is achieved by linking FIX DMACS database blocks to areas in the memory of the low level device controller. The third way in which information in the VMD can be altered is by the intervention of the Device Control Algorithm (DCA) which is based in FIX DMACS. Further information about the operation of the VMD can be found in appendix A.5.

### 5.2.2.3. Device Control Algorithm

The device control algorithm is an algorithm that has been implemented within FIX DMACS to control the actions of the low level device. Its operation within a cell has been tested though manual simulation of the VMD.

The Device Control Algorithm (DCA) makes decisions based upon the current state of the virtual model (VMD). The algorithm was designed to be generic in nature, thus it was able to be applied to all different types of low level devices. The logic of

the DCA was defined as a series of steps which would be common to all devices. The steps to the algorithm are shown in figure 5.3.

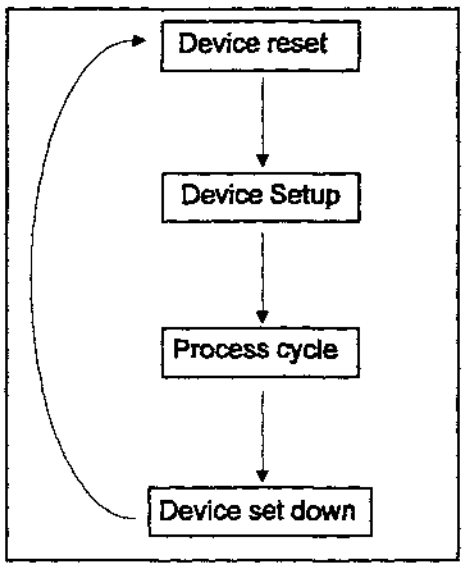



Figure 5.3
Four Steps of device control algorithm

The four steps of the DCA are what the algorithm will follow each time it runs through a cycle. Each step in turn contains a variety of sub steps that run to accomplish each task designated to be in that step. The DCA was implemented using programmable blocks in the FIX DMACS database. These program blocks (FIX DMACS Software Manual 1993) as they are called, are capable of containing a short user defined program. The language which is used to define the algorithm, allows a variety of functions to be performed on the information contained within blocks in the FIX DMACS database. These program blocks did however have a number of limitations (For further information see appendix A.6), the most restricting of these was the limit of 19 instructions per block. This meant that to implement a large program a number of blocks were required to be linked together.

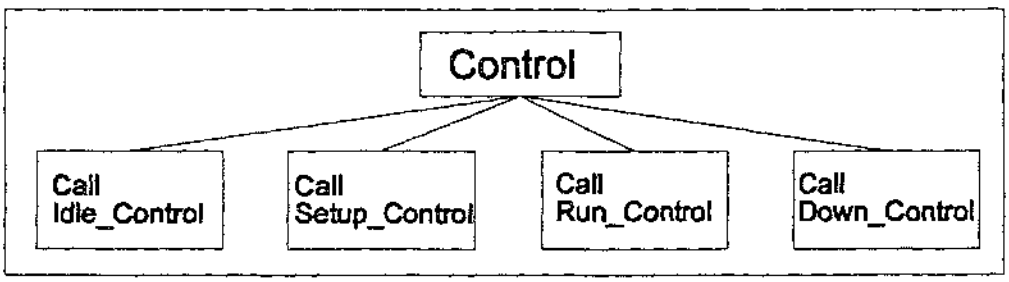



Figure 5.4
Master control algorithm

Figure 5.4 demonstrates the master control algorithm that calls other program blocks, these secondary program blocks represent each of the four steps, defined as the four major steps in the device control algorithm represented in figure 5.3. Appendix C provides diagrams that define the algorithms that the device control algorithm consists

of. The program blocks operated by reading information from the VMD then making decisions based on this information, then altering the VMD to reflect its new state.

*Step 1, Device Reset*

This initial step is designed to wait until a job appears in the "to be processed queue" or as it is called the current queue. This is the buffer where jobs to be processed for a particular device are stored.

1. When a job appears in the current queue, first the program block resets the VMD and all associated database blocks into an initialisation state. This also prevents any spurious errors which may have occurred since the last cycle effecting the current one.

2. In the next stage of the Device Reset algorithm, the low level device is run through any INITIALISATION procedures that are required before it can begin its cycle.

3. Finally the low level device is checked to ensure that these previous steps have not caused any errors. If this occurs then an error message is sent to the operator and the device control algorithm halts.

*Step 2, Device Set up*

This step consists mainly of retrieving information to enable the next phase of the operation to take place. This process is demonstrated in figure 5.5. First the next job to be processed is POPPED off the Current Queue, it then becomes the current job. The current job information is placed into the respective VMD via DDE from file by an application called Next_Queue. Next_Queue is a Visual Basic application that contains both the incoming "Current_Queue" and outgoing "Next_Queue" queues for each VMD.

The Next_Queue application retrieves the current task from the file (Job Record data object group) specified by the Device Set up algorithm then transports the appropriate information into the VMD. This gives the device control algorithm the necessary information to process the current task.

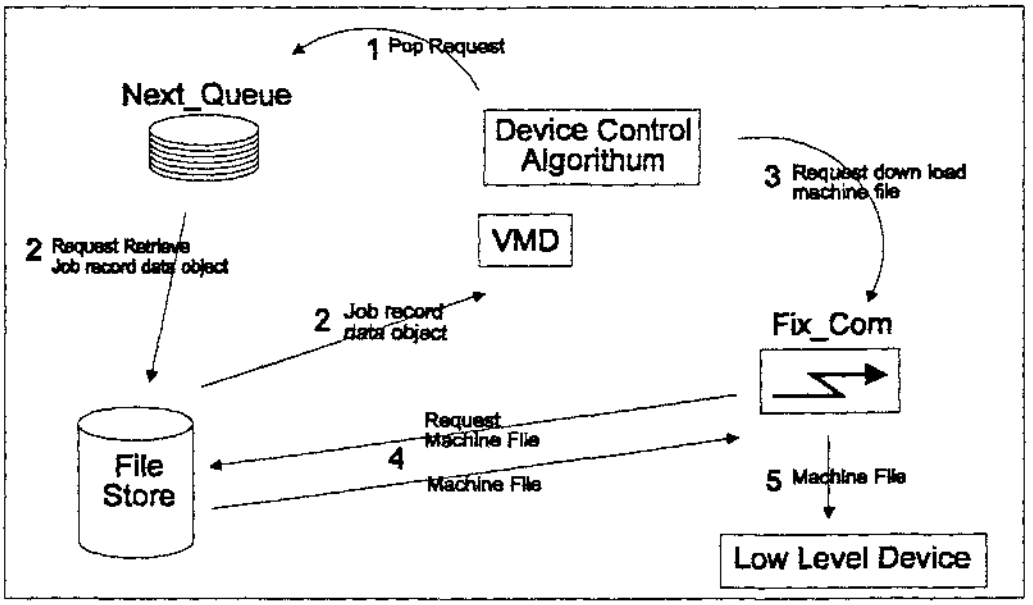



Figure 5.5
Device Set up Process

The VMD will now contain information regarding the name of the Machine data file, the communication parameters, destination and type of file. The Device set up algorithm then communicates with another Visual Basic program called FIX_COM to undertake the transmission of the machine data file (Cutter Location File) to the respective low level device. FIX_COM then retrieves the appropriate file from the network server and transmits it to the low level device via the appropriate communications port specified in the VMD.

*Step 3, Process Cycle*

The process cycle is the most complicated of all the steps in the DCA. This cycle is responsible for the actual operation of the low level device in order to complete the task at hand.

1. The first step in this algorithm is to check that the low level device is in a state to continue, if it is then a command is sent to LOAD the work piece into the low level device. The algorithm then waits until it receives a LOAD_COMPLETE signal, which tells the DCA that the loading of the work piece is complete. The low level device is checked for an error condition, if at any stage during the operation of the device control algorithm an error is detected and the low level device is unable to continue then a partial rebid program will operate.

2. The next step for the process cycle is to START the low level device processing the current work piece. The processing of the work piece is then monitored and has the capability to check any error conditions. When the CYCLE_COMPLETE signal is received, the low level device is UNLOADed.

3. Process timers give the time taken for the current cycle, the total number of cycles to undertake, the number of cycles left to complete, the time taken so far in processing the Job and the time left to process. The time left to process is only an estimated time, which is initially based upon an estimate that was provided in the task information (Job Record data object group), however as the job is being processed then the estimate is updated and the time left to process is re-calculated to give a more accurate estimate. This is done through using the latest cycle time as the new estimate for each consecutive cycle left to complete.

*Step 4, Device Set-down*

In this step any operations that are required to return the low level device to an IDLE state are undertaken. At this stage the current task in the job has been completed and the next task in the job now requires to be processed. The device set-down step takes the next task and ensures it is placed back into the system to be completed. It does this by instructing the Next_Queue application to PUSH the Next Queue data object group in the outgoing/next queue. When appropriate the auction control entity will then hold an auction to release this job so the next task can be completed.

### 5.2.2.4. The Auction Control Entity

A bidding process was devised to maximise the autonomy of each of the cells involved in the auction. This process allows each cell within the factory to take part in a negotiation process in which enabled the distribution of jobs amongst the capable cells within the factory. When implementing the Auction based system it was realised that the effectiveness (Shaw 1987) of the bidding approach would be determined by the quality of the information available to be used to formulate the bid. The quality of the information can be determined by a number of factors including; timeliness, number of parameters, availability and type of information.

Therefore the auctioning system was best implemented where quality information could be obtained about the state of the low level device, FIX DMACS. Data flow diagrams for each of the different entities that perform the auction process are shown in Appendix D. The functioning of the Auction control was tested extensively using one complete cell controller, one complete factory controller and two simulated cell controllers.

The auctioning system consists of an interaction between two major entities the auction formulation or control entity within the cell controller and the auction decision entity within the factory controller. The auction control entity is the entity that calculates the bid that a cell will place with the aim of winning the auction. The auction decision entity makes the decision on which cell will win the auction.

The proposed auction based system used FIX DMACS network capabilities to great effect. The auction control entity consisted of three major elements bid formulation, re-bid control and auction co-ordination. The bid formulation (Simplified diagram of Bid Formulation algorithm in figure 5.6) element was the part of the cell controller that calculated the value of the bid in response to a request. This was done through the use of Program blocks and Calculation blocks within FIX DMACS. The program block provided the control of the different steps in the bid formulation process and the calculation blocks provided the actual calculation of the bid. Further information about the following elements of the auction control entity is available in appendix A.7.

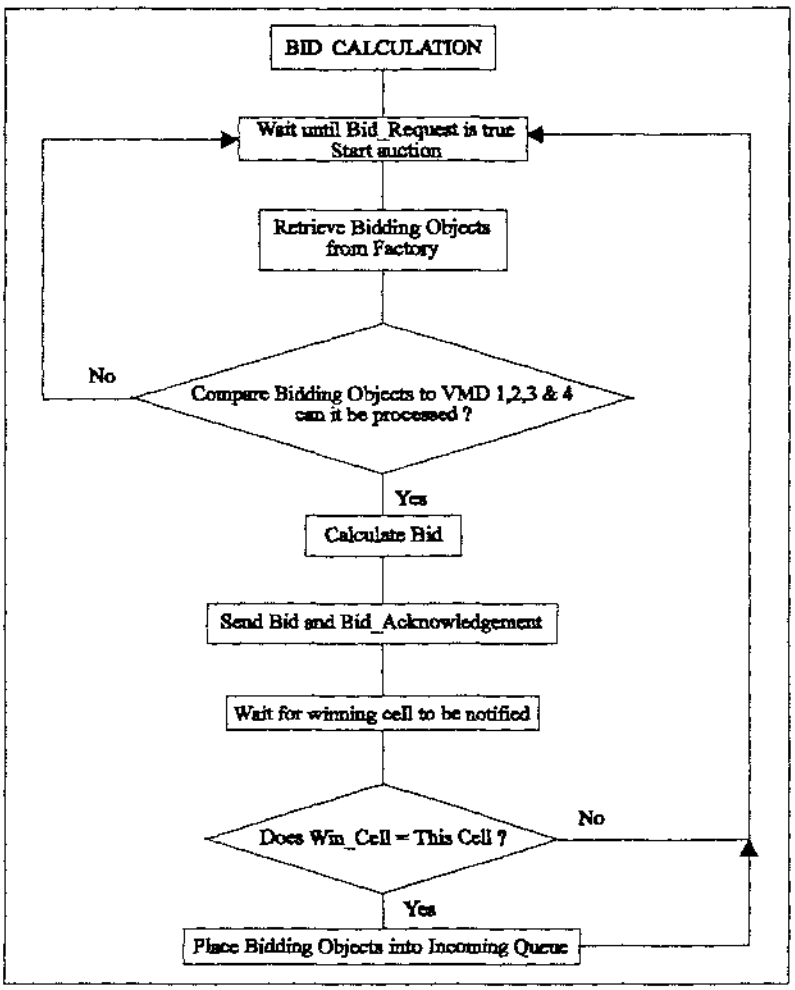



Figure 5.6
Bid formulation algorithm

*Bid formulation*

1. The bid formulation or bid calculation occurs at the cell controller in response to a BID_REQUEST. A bid request is generated by either the factory controller releasing a new job into the system or by a cell re-bidding a partially completed job into the system, in either case a BID_REQUEST is received by the cell controller and the bid formulation process begins.

2. Next the cell controller receives the Bidding information relating to the job at the centre of the auction, via the LAN from the factory controller. The details of this transfer of information can be shown in Appendix E as the "Bidding data object mapping from cell controllers via the view screen".

3. Once the bidding objects have been received, the next step is to decide if the cell is capable of undertaking the task at the centre of the auction ie the auction task. It determines this by comparing information about the auction task, ie Bidding information (data object group) with the capability's description of the low level devices, ie the VMDs. This process first checks to see if the cell has a low level device capable of performing the task, it then checks to see if that device has the necessary resources to undertake the auction task.

4. A calculation block is then used to calculate a bid. The bidding calculation is as follows:

$$\begin{pmatrix} Current\_QueueTime + (No.\_of\_Cycles \times (CycleTime + \\ (LoadTime + UnloadTime)) + SetupTime) \end{pmatrix}$$

This assessment of the Earliest Finishing Time (EFT) is then transported back to the factory controller. Once the factory controller has collected the bids it can then make a decision on which cell will be awarded the job. This decision is based upon earliest EFT. When the winning cell has been informed that it has been awarded the job at the centre of the auction, it then retrieves the current queue data object group, from its VMD then PUSH's it on to the current queue.

*Re-bid Control*

The re-bid control function has the task of releasing partially completed jobs back into the system to be completed. This can occur when either a DCA has finished processing a task, or when a low level device breaks down, in both cases the current job requires rebidding.

In the case where the current task has been completed, the next task from its job file must then be re-bid. If there is a next task that requires re-bidding then the bidding objects are retrieved from the appropriate job file (IFFile). The bidding objects are retrieved by the Next_Queue application from the task which next to be processed.

1. The re-bid control algorithm holds in a wait state until it is polled by the bid coordinating entity. Whilst the cell has the bidding token it can request an auction.

2. As soon as the re-bid control algorithm receives the bid token it then checks each of the four next queues (one next queue for each low level device) to see if they have any tasks that require auctioning.

3. If a device has a task that requires to be auctioned the re-bid control algorithm then takes the bid token and the auction process begins. Figure 5.7 demonstrates the

auction message passing process from the auction decision entity at the factory controller to the auction control entity. Time moves from the top to the bottom of the diagram.

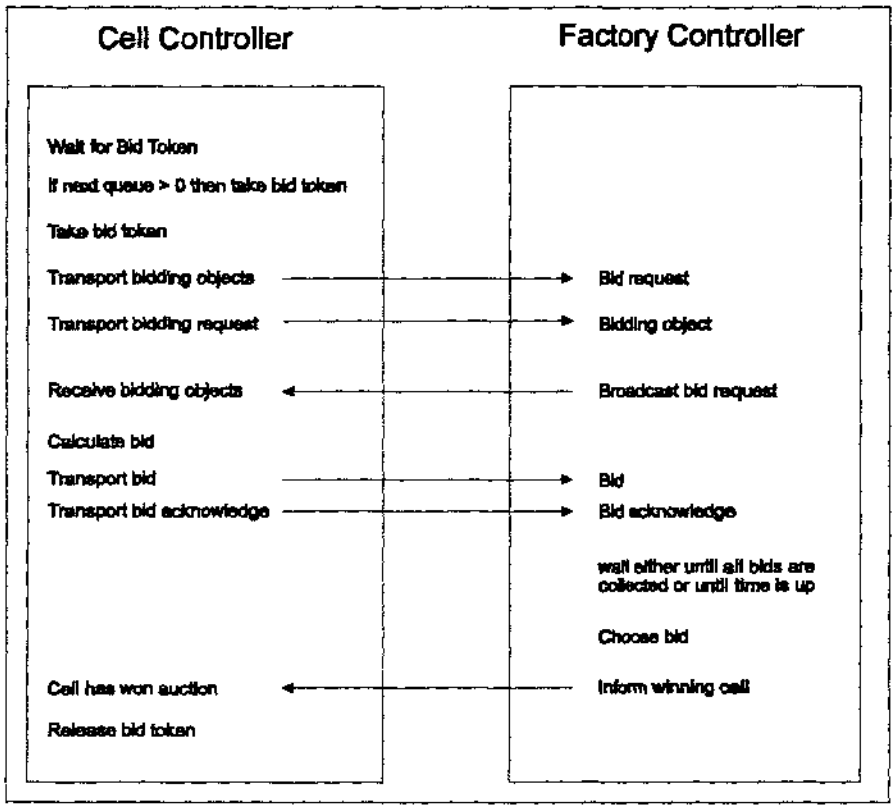



Figure 5.7
Auction message passing

4. The Bid_Request and the Bidding Objects are then sent to the factory controller. This occurs when the Bidding Objects are popped off the Next Queue. These Bidding Objects are then sent directly to the factory controller and from then are broadcast upon the network to all other cell controllers.

*Auction Co-ordination*

The auction co-ordination entity is responsible for the polling of the factory controller and cell controllers. This polling serves as a method of limiting the number of entities who are currently undertaking an auction. Due to the way in which the auction process has been implemented using FIX DMACS, it is possible to only hold one auction at once. This was achieved through the creation of an application written in Visual Basic. This application was fully implemented and tested in co-operation with a partially completed cell and factory controllers. Further information about the auction co-ordination entity is available in appendix A.8.

An application was written called Bid_Rotate. This application had the task of moving a bid token around all the capable entities within the network. Bid_Rotate operated, through using two DDE links to a database. The DDE links used by the Bid_Rotation application are shown in Appendix E as Bidding token data objects mapping's. These two links are rotated around the factory controller and each cell controller in turn. Effectively shifting the focus of the Bid_Rotation application around the factory one bidding entity at a time. The first of these two links, the Bid_Token link is a data object that an auction entity must have in order to initiate an auction. The second link is the Bid_Operate link this a data object which a bidding entity must have in order to halt the rotation of the bid token. Thus if a cell wants the bid token it must wait for the Bid_Token Bidding Object then set the Bid_Operate block too high. The program listing for the Bid_Rotation program is available in appendix A.1.

#### 5.2.2.5. Bidding Information Transportation Entity

The bidding transportation entity is the entity responsible for the transportation of bidding objects. This transportation of bidding objects consists of two separate transportation functions, to and from the cell controller. These bidding objects have to be transported from the originator of the auction, ie cell controller to the auction decision entity at the factory controller, shown as case 1 figure 5.8. Bidding objects have also to be transported from the factory controller to all the cell controllers, shown in case 2 figure 5.8. These two different transportation functions have been undertaken by different applications within the cell controller. Both of these bidding transportation functions have been implemented one from within FIX DMACS and the other within a Visual Basic application. The operation of the Bidding information transportation entity has been tested using an experimental factory consisting of two cell controllers and a factory controller. Further information on the operation of the Bidding Information transportation entity is available in appendix A.6.

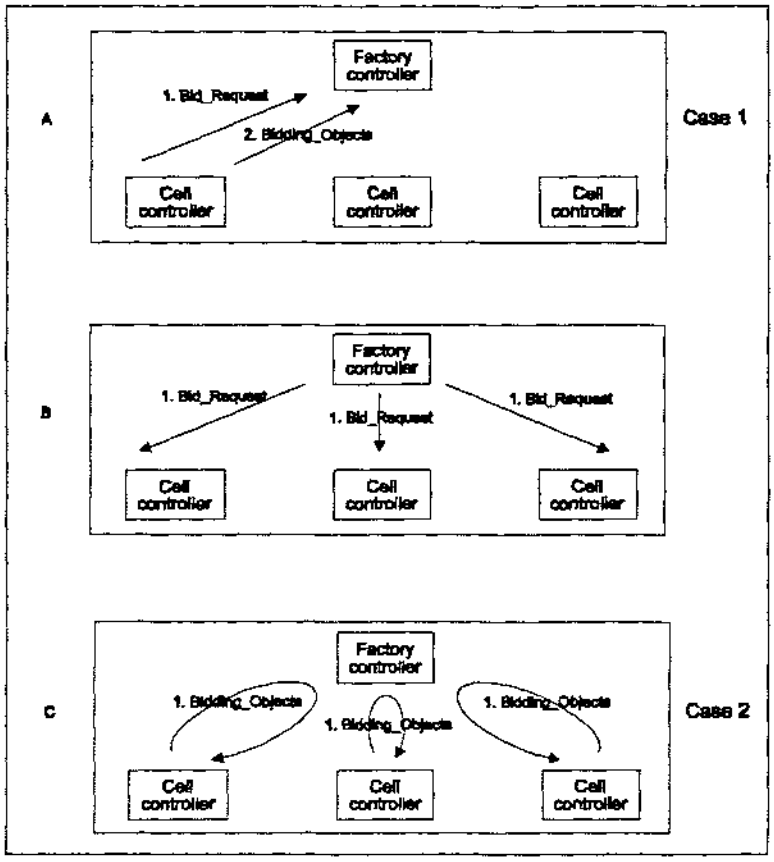



Figure 5.8
Three initial phases of communication in the auction process

The first case of the Bidding Object transportation is easily undertaken through using a Visual Basic application called Next_Queue. The Next_Queue application contains a queue called Next Queue. This queue stores the Bidding Objects from tasks waiting to be re-bid. When the Re-Bid control entity decides to Pop a set of Bidding Objects off this queue, they are automatically sent along with a Bid_Request via DDE to the factory controller.

The second bidding transportation function is the transportation of Bidding objects and the Bid Request, from the factory controller to all of the cell controllers on the network. This function has been implemented within FIX DMACS at the cell controller. A FIX DMACS macro was written within the provided macro editor application and attached to a view screen running on each cell controller. This macro transported all the bidding objects from the factory controller to the cell controller via the network. This occurred when the Bid_Request object in the factory controller was set high. The transportation of the actual Bid calculated by the cell and the subsequent Bid_Acknowledgment is also transported by the macro attached to the cell controller view screen. (Further information about the Bidding Information Transportation Entity can be found in appendix A.9)

### 5.2.2.6. Incoming and Outgoing Queue Entities

The incoming and outgoing queues are buffers for each low level device in every cell. This enables the DCA to be decoupled from the auctioning and re-bidding of jobs. Both of these buffers or queues (current or incoming and Next or outgoing) have been implemented within a Visual Basic application called Next_Queue. They are able to communicate to a FIX DMACS database via a number of DDE links, see Appendix E for list of DDE links for the Next_Queue application, they are defined as the Next and Current Queue data object group. The application that contains the two queues has been successfully tested both via simulation and in conjunction with an operating cell controller.

The incoming queue or current queue stores at each item in the queue the name of the IF File and the name of the associated job. An item is placed into the current queue when a particular low level device has bid for a task and won it. Items are removed from the current queue when a low level device is ready to process a new task. Each item in the next queue contains a number of fields, these correspond to the next queue data object group which are designed to store Bidding Objects. These fields must contain all the information required to re-bid the task in the current job. Items are placed on to the next queue when a particular task in a job has been finished and the next task requires auctioning. Items are removed from the next queue when the auction control entity at the cell controller is ready to hold an auction. The item at the top of the next queue is then removed and auctioned.

The queue itself consists of a number of items placed in a sequential order, a number of operations can be performed upon the items within the queue. The items within the queue can either be retrieved from the top of the queue (POPed) or items can be placed at the start or end of the queue (PUSHed). The queue operates on a first-in-first out basis.

Like all the Visual Basic applications that make up the cell controller, Next_Queue operates as a back ground task. Program code and window screen dumps for the Next_Queue application is provided in Appendix F. Further information on the incoming and outgoing queue entities is available in appendix A.10.

### 5.2.2.7. Control Transportation Entity

The control transportation entity is the function of the cell controller responsible for the transfer of information to and from the VMD and the Intelligent Entity. This entity must interact with both the IE and the FIX DMACS database. This entity also has to be capable of performing this information transportation task with IEs that consist of different programmable devices. The functioning of the control transportation entity is undertaken by a software application called a FIX DMACS I/O driver. This software operates with FIX DMACS and provides the link between FIX DMACS database input and output blocks and the locations in the memory of the programmable device. A variety of I/O drivers are available for most well known PLCs and can be obtained from the makers of FIX DMACS.

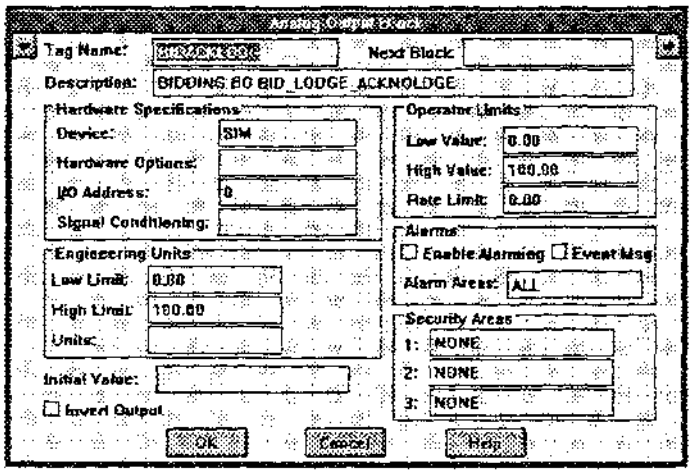



Figure 5.9
Specification of a FIX DMACS Analog Output database block

Input and output blocks can then be linked to via the I/O driver to addresses in memory of the PLC, by entering the appropriate address during the specification of the database block, as shown in figure 5.9, (I/O Address). The I/O driver then scans specified addresses in the PLCs memory and transfers the information to the database block. Conversely it scans the value of the database block and then writes this value to the specified address within the PLCs memory (refer FIX DMACS manual). Unfortunately an I/O driver for the Fisher and Paykel PSC, which was the type of PLC chosen for the first test implementation had not yet been completed.

### 5.2.2.8. The Data Transportation Entity

The data transportation entity is the entity at the cell controller that transfers the machine data file at the specified communications parameters down to the low level device. This is an operation which was required to be implemented within a Visual Basic application, this application called FIX_COM. This application has been completed and tested using a simulated low level device controller. The machine data file, is a file that contains instructions that pertain to a specific type of low level device, which tell the low level device's controller how to manufacture the work piece. The file usually consists of G and M code. It also provides a list of co-ordinates and instructions that tell the low level device explicitly how to manufacture the work piece at hand. The machine data file is usually created by the operator at the factory controller.

Unfortunately due to the opportunistic scheduling method one can never be certain where a specific task will end up to be processed. Therefore it was necessary to produce generic machine data files and then give each cell the ability to post process the generic code into a form which the specific low level device can interpret.

The task of the data transportation entity is to take the machine data file after it has been post-processed by the cell and to transfer it via a communication port available on the computer to the low level device concerned. For this task an application was written in Visual Basic, this application interfaced within FIX DMACS via DDE links. These links enabled the DCA to control the operation of this application. Further information on the specific operation of the Data transportation entity can be found in appendix A.11. The program code for the visual basic application is available in Appendix G.

### 5.2.2.9. Monitoring Entity

The monitoring entity is the entity that provides visual feed back from factory to the operator. This enables the progress of the devices in the cell to be monitored and lets the operator view specific information about any of the devices within the cell. Due to its supervisory control capabilities FIX DMACS was the most suitable platform for which to implement this entity. Through using a number of FIX DMACS View screens linked together it was possible for a visual representation of the current state of the cell to be created. The view screens within FIX DMACS also had the tremendous advantage of being able to access information on any cell controller from one cell.

This entity has been partially implemented. It was felt that a number of detailed information presentation tasks were beyond the capabilities of FIX DMACS, whose historical information representation abilities were somewhat limited. Thus a combination of FIX DMACS view screens and Visual Basic applications would be required, these extra Visual Basic applications have as yet only been partially completed.

At the cell controller level the information displayed to the user was divided into three different levels, as shown if figure 5.10.

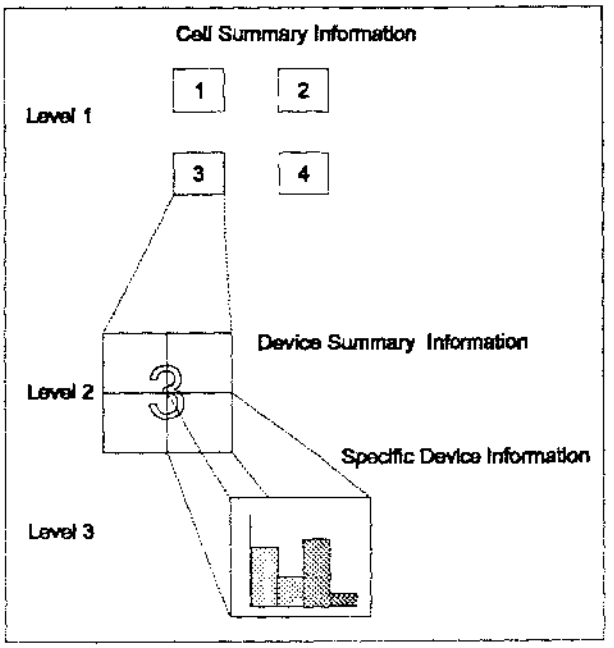



Figure 5.10
Three levels of information representation

These three levels of information, represent three levels of increasing detail as the operator travels down the levels. The first level represents general summary information regarding the devices in the cell and what state they are in, as shown in figure 5.11.

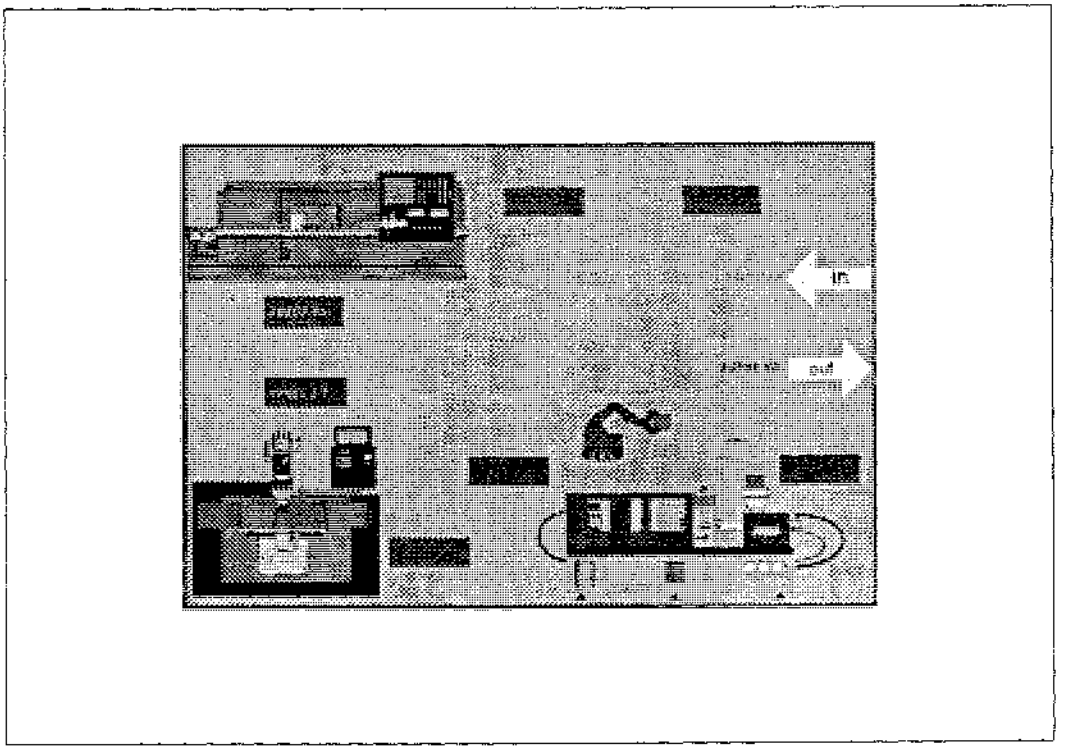


Figure 5.11
A typical FIX DMACS view screen of a cell controller

The first level was completed entirely within FIX DMACS, as it requires only current state information from the local VMDs. This first level of view screen represents the physical configuration of the local cell.

The second level provides more detailed information, but this time only about a specific device. The user is able to select this more detailed information by using the pointing device on the computer screen to select one of the graphics representing a low level device. Selecting one of the up to four graphics (cells) on the screen causes a macro to load the appropriate next level down view screen. This view screen gives a more detailed representation of the current state of the low level device. At this level the on screen representation of the low level devices actually alter dynamically as the actual state of the real device changes. This second level of information representation also provided detailed information about the state of the current and next queue, ie number of jobs in the queue, the name of the job at the top of the queue and the queue length in seconds.

The third level of information available to the operator at a cell, is the most complicated of the three levels and has been only partially implemented. This level uses both FIX DMACS view screens and Visual Basic applications currently being written, to provide detailed information on specific areas. The user will be able to gain access to this information through using the pointing device or button that will take the operator through in to the appropriate screen. The information available at this level will most likely be, device performance over time, device utilisation, jobs undertaken, estimate production times vs real production times and forecasted utilisation.

### 5.2.3. Implementation of Low Level control, Intelligent Entity Layer

The Intelligent entity is the lowest level of control in the flexible manufacturing system. This level of control takes the generic commands issued by the cell controller via the VMD converts them into application specific commands and executes them via the PLC type device. The implementation itself consisted of three major parts, the link between the cell controller and the IE (I/O driver), the implementation within the PLC type device itself and the connection to the low level device. The Fisher & Paykel PSC was used to gain maximum flexibility for the test implementation. The implementation of the IE within the PSC consisted of three parts, the communications with the cell controller via the I/O driver, the generic decision rules and the specific implementation. These parts being;

1. The VMD. Each database block within the VMD at the cell controller will have a corresponding address in the memory of the PSC. Any information passed into an output database block in the VMD at the cell controller will be mapped onto the address in the memory in the PSC.

2. Generic Decision Rules. These are a series of programs within the PSC that interpret the generic commands given to the PSC then decompose these generic commands into a series of instructions. At this stage the PSC may have to make

decisions about how best to undertake the command issued by the cell controller, based upon information about the low level device. For example a command to load a lathe within a cell is received by the PSC from the cell controller, the PSC may decide however to wait a few seconds until the automatic loader is ready, and then decompose the command into a series of instructions.

The rules at this level are simple and the decision making is basic and exists mainly to take advantage of the timely information available at the PSC. The decision rules are required to be defined in a generic terminology so as to be applied to the implementation of decision rules in any type of programmable device. These rules have yet to be formulated as more research is required into this area.

Figure 5.12 demonstrates how the VMD and the generic decision rules interact to implement a command or request for data.

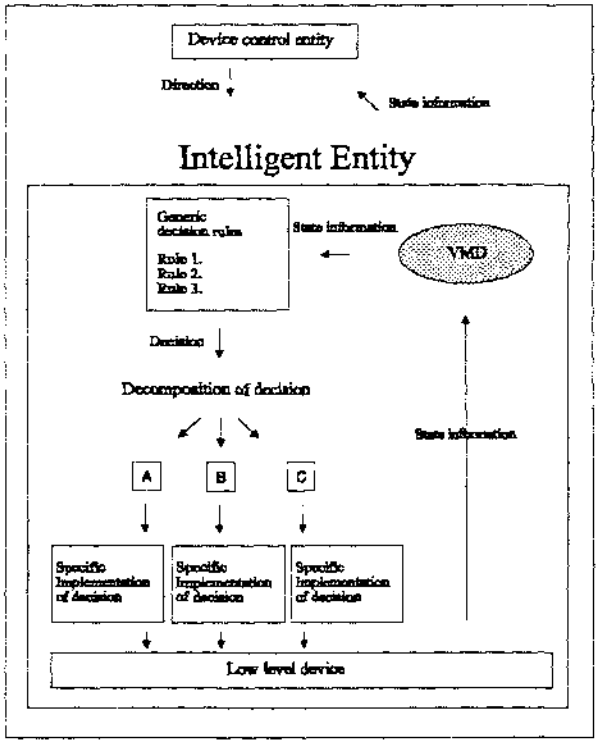



Figure 5.12
Interaction between generic decision rules and the VMD

When these generic decision rules have been implemented they should function the same, no matter on which PLC type device they have been programmed upon. This method adopted for generic message decomposition has the aim of lowering the specific implementation down to the lowest possible level to allow new devices to be added or reconfigured with minimum effort.

3. Specific Implementation Routines. These take the decomposed decision from the generic decision rules and implement each individual instruction in the physical system. This layer also uses sensors connected to the physical device to gain current state information and uses this to update the VMD. The programming and

functioning of these routines will differ between types of programmable devices and between different physical devices. The implementation of the decomposed decision will require the programmable device, in this case the PSC, to take the instructions one at a time and effect them upon the physical device through using the sensors and actuators available that are connected to the physical device. For an example of instruction decomposition refer to appendix A.12.

As far as the implementation of this has gone, a number of routines have been written within the PSC in its state programming language to undertake the specific implementation of the generic commands. Furthermore a number of routines have been written to undertake a variety of I/O operations and tasks within the PSC itself and can be utilised in the future by the specific implementation routines, these are included in Appendix H, together with the functional diagrams.

Two low level devices were available in the Department of Production Technology for research work, a Bridgeport series 1, Interact mill and a Taskisawa LS5 Lathe with a TS-15 CNC controller. Research had already been conducted on these two devices. A RS-232 link between a computer and both of the low level devices has be established, with a number of machine data files being down loaded and subsequently run. From this it is possible to establish what portions of both devices could be controlled via the on board controller and what functions would be required to be hard wired from the PSC, in order to automate the low level device. This meant that from a combination of commands transferred via the RS-232 port of either the cell controller or the PSC itself and changes made to the control panel of the device via the PSC's switches and relays, it is possible to completely control the functioning of both low level devices. Sensors to be utilised by the PSC will enable information regarding the current state of the low level device be gained. Further information about the suggested implementation of the model shown in figure 3.7 can be seen in appendix A.8.

### 5.2.4. Implementation of Factory Control Layer

The factory controller is a single FIX DMACS node consisting of one database with additional Visual Basic software. The factory controller exists to provide some of the high level management of the factory and also to allow a centralised access point to a number of crucial functions, such as CAD/CAM, simulation and job definition. As yet not all the functions of the Factory controller have yet been identified for the formulation of the model, thus the list of entities which undertake these functions should be treated as inclusive.

Visual Basic was utilised to implement a variety of different functions these being; information management, factory configuration, job definition, job release queue, factory management and transportation functions. As yet the engineering design, tooling and fixture's management, master production schedular and the factory simulation have not been implemented.

FIX DMACS was utilised at the cell control layer to provide network communication facilities as well as provide the implementation of the auction control entity and portions of the factory management entity.

### 5.2.4.1. The Factory Controller

The factory controller currently consists of three major parts integrated into one environment by Visual Basic application called Virtual Factory. This is the front end program that leads the user through the various functions of the factory controller. A number of functions are unable to be undertaken by this application, such as factory monitoring, network communication, auction decision making and engineering design functions. These functions had to be undertaken by other applications. In the case of the engineering design functions CAD and CAM package are used, these applications were accessed from the virtual factory application. The factory monitoring functions and the network communications are undertaken by FIX DMACS. FIX DMACS sits as a background task providing network communications and auction control functions whilst the Virtual Factory application is operating. When the user wishes to monitor the progress of the factory, FIX DMACS is brought into the foreground and a number of view screens are opened providing up to date information on the current state of the factory.

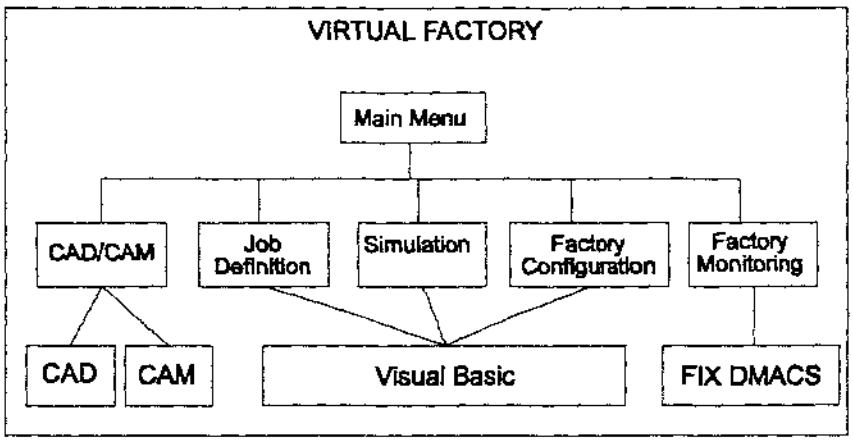



Figure 5.13
Structure of applications within the factory controller

Figure 5.13 demonstrates the integration abilities of the Visual Basic application, Virtual Factory. This application selects which ever application is capable of undertaking the task that the user has requested. Virtual Factory consists of a number of different functions integrated into the one package. The Virtual Factory application is menu driven eg main menu is shown in Appendix J. The choices that the user can make are to move into one of the functional areas represented in part by figure 5.13, these being Simulation, System monitoring, System definition, Reporting, System definition and Job Definition. The operation of the entities that undertake these functional areas of the Virtual Factory application are discussed in following.

### 5.2.4.2. Implementation of the Factory Configuration Entity

The factory configuration entity is the entity which allows the operator to alter the state of the virtual factory. This entity is part of the Virtual Factory main program. This functions by allowing the operator to define the configuration of a virtual factories set up, the low level devices available as well as what tools and fixtures are available. Of these operations only the first two have been fully implemented.

From the main menu within the Virtual Factory program if the operator selects the option to configure the factory (System Definition option) then a new window is open that contains another four options for which to choose from, ie figure 5.14.

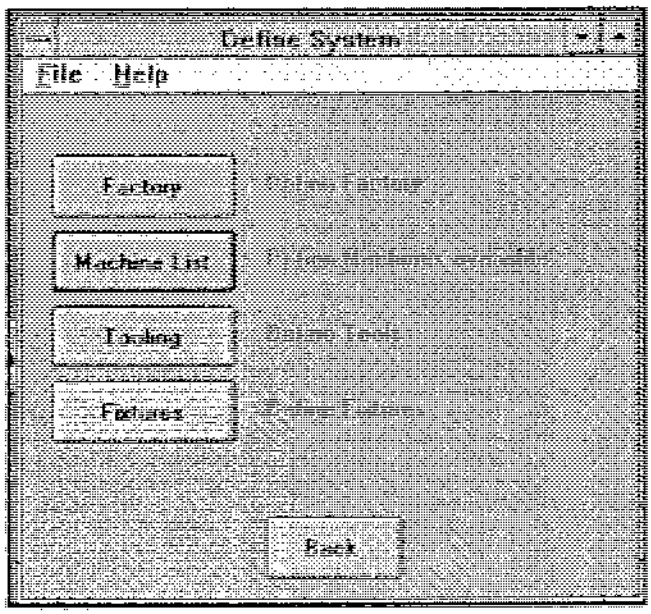



Figure 5.14
Factory configuration main menu window

The first button in this window allows the user to configure the structure of the virtual factory. This then takes the operator into a further window figure 5.15 where they can select to either configure the layout of the cells in the factory or define the number of cells in the factory. Further information on the definition of the factory cells is available in appendix A.14.

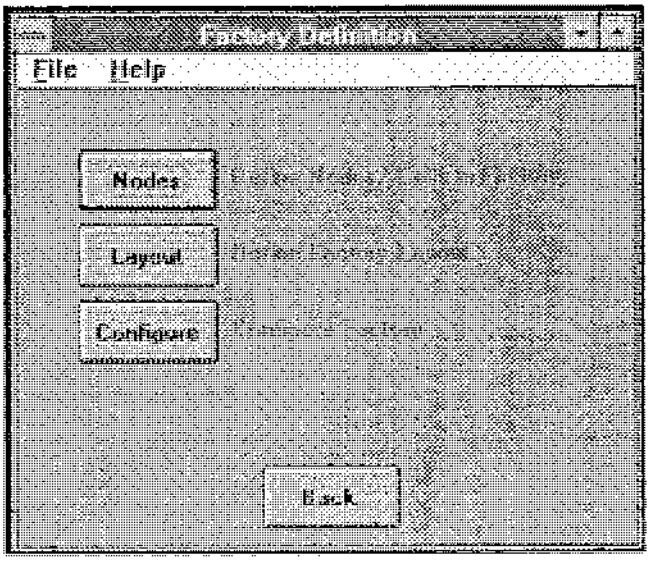



Figure 5.15
Factory definition menu window

Once the user has defined the cells that the factory consists of, the operator is then able to define the configuration of each of the devices in these cells. This is done through selecting the layout option from the menu shown in figure 5.15. This then takes the user into a screen which allows them to select which cell then would like to define the layout. Next the user is then able to define the actual virtual configuration of each device in that cell. This consists of entering information such as device type, name, set up time, down time and so forth. All this information is written to a file, this file is then used to transfer this information to all the VMDs in all cells, thereby configuring the factory as defined by the user. Further information about process of defining the layout of the factory is available in appendix A.15. The structure of the Virtual Factory should of course reflect the actual structure of the real factory.

Selecting the machine list from the menu opens a further window called the machine list. This window (Figure 5.16) allows the user to define a list of machines or low level devices that are available to be used in the factory definition. Therefore when the user is defining the factory layout, a list of machines will be available for which they can select from. In an operational factory this list will contain all the low level devices available in the factory.

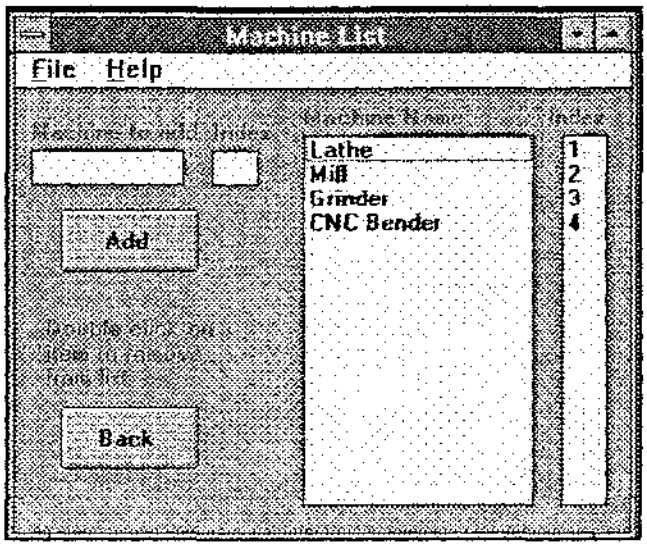



Figure 5.16
Machine list definition window

Once the user has defined all the machines in the factory and has allocated individual indexes to them all, the user can then return to the factory configuration main menu. When the user returns to the main menu the list of machines available to the Virtual Factory is stored within a file for later use. This file is called the "machinlt.mch" file and contains an up to date list of all devices in the factory.

### 5.2.4.3. Implementation of the Job Definition Entity

The job definition entity is the entity within the Virtual Factory application that allows the user/operator to define the contents of a job to be processed. This entity is also responsible for the information management of the job files it produces. This entity has been fully implemented within the virtual factory application.

The job definition entity can be accessed from the main menu of Virtual Factory. By selecting the job definition option from the main menu the user is presented with a window, Appendix J, which allows the user to either open a new job file (IF File) or open an existing file. Once a file has been opened the user is then offered another window, figure 5.17.

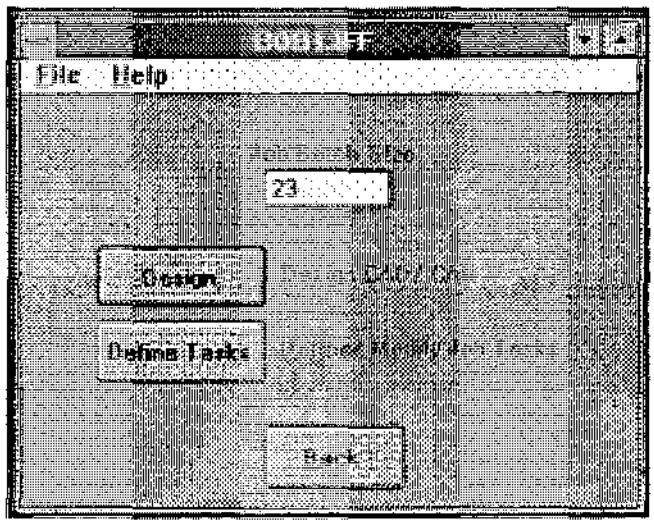



Figure 5.17
Job batch size window

Figure 5.17 shows a job file (bob1.iff) has been opened, the batch size has been defined as 23. The user next has the choice to design and produce machine data files, by selecting the design CAD/CAM option or to define the tasks that the job consists of. The link between the CAD and CAM packages has not yet been completed. However, selecting the Define Tasks option will cause the job definition window to open. This window, shown in Appendix J, allows the user to enter all the information pertaining to a particular job.

The definition of a job consists of specifying the information pertaining to each task in the job. In the task definition window all information relating to a specific task is entered. This process is repeated for each task in the job, the order in which the tasks are defined is the order in which they will be processed.

The task information consists of a description of the time required to process the job, a list of the resources required and a number of fields where file names can be entered so data files such as the machine data file can be linked to the task. Once the first task has been defined the user can then move onto the next task by selecting the movement control. This process is repeated until all the tasks within the job have been defined, at this point the user can then save all this information into the IFFile. Once the job has been fully defined it can then be placed upon the job release queue, this is a queue which the job will stay in until its data to manufacture (entered by the user) becomes the current date.

### 5.2.4.4. Job Release Queue and Master Schedular Entity

The job release queue consists of two queues, the first being the job release queue and the second being the master queue. The job release queue is simply a list of all the jobs to be produced and at what date they should be started. The job release queue consists of a file containing a list of all the jobs to be completed and the respective start dates. Every 5 minutes this file is loaded by the job release entity, the list is

searched to see if any jobs have a start date that is either today's date or earlier. If so then this job is placed into the outgoing or master queue and will from there be processed.

The contents of the job release queue can be viewed from the factory monitoring function. From the factory management main menu the user can choose the monitor jobs function, this will provide the user with a list of all jobs and manufacturing start dates currently on the job release queue. From this point if the user chooses a job that is not due to be started yet can either be forced onto the master queue or can be deleted. This enables rush jobs to be completed or jobs to be cancelled. This entity has been entirely implemented within the virtual factory application.

The master schedular entity is the entity responsible with interfacing with the job release queue, deciding what resources are required and then scheduling the supply of these resources. This entity has not yet been designed or implemented.

#### 5.2.4.5. Implementation of the Factory Management Entity

The factory management function or factory monitoring function is the task within the virtual factory application that allows the operator to view the current state of the factory. By selecting the factory monitor button from the main menu, a window is opened which allows the user to choose between, viewing the job release queue, viewing the master queue or viewing the factory itself. When selecting the option to view the factory the virtual factory application opens a FIX DMACS view screen similar to the ones found at the cell controllers. In the case of the factory controller however, the information available to the operator can be obtained from any one of the cells on the network. This entity has not yet been entirely implemented. Most of the view screens have been created, however they have not yet been linked properly to other databases in a fully operating factory.

The first view screen that the operator is provided with at this level is a summary of the entire factory. The operator can gain further information about any one of the cells by selecting one of the icons relating to one of the cells in the factory. This then provides the operator with a screen similar to the first layer screen supplied to the operator at the cell controller, at this stage further information can be gathered about any of the devices in the cell by gain selecting one of the icons that represent one of the devices. From the factory controller it is not possible to "burrow" down through the different levels of increasing focus and information complexity to the same level as with the cell controller with respect to gaining information about any one device. It is possible to gain relatively detailed summary information about groups of devices within the cell. The FIX DMACS view screens currently completed, which make up the factory controller monitoring functions are shown in Appendix J.

#### 5.2.4.6. Implementation of Auction Control Entity

The auction control entity has three main functions, it firstly transports the bid request that arrives from a rebid entity any where within the factory to the cell controllers within the factory. Its second function is to undertake the task of rebidding jobs in the

master queue. Thirdly it makes the decision on which cell will win an auction based upon the bids it has received. These functions of the auction control entity are all implemented within FIX DMACS and have been created using program blocks within the factory controller's data base. The rebid control function at the factory controller acts identically to the rebid control entity at the cell controller.

The bid request transportation function is a simple one, when a bid request is received from any entity in the factory this bid request is transported via a macro written on a factory controller view screen. This macro, which can be found in all the view screens at the factory controller, takes the bid request received from any cell controller or the factory controller and transports it to all the other cells. It does this by setting all the "bidrequest" database blocks in all the cell controllers to 1.

The program block that decides which cell will win the auction, is called "winbid". This program block waits until a Bid request is broadcast upon the network. It then starts a timer, and waits for bids to be placed by each cell. When either the cell controller receives bids from all the cells in the factory or the timer is finished, the winbid program block closes the auction. At this point it then makes a decision on which of the cells will win the auction, it does this by simply comparing all the bids and selecting the smallest one. The smallest bid relates to the cell that can undertake the task in the shortest period of time, ie earliest finishing time. Once this has been done the winbid program block notifies the winning cell that it has won the auction. This is done by setting a database block in the winning cell to 1, this block is called the "winbid" block.

### 5.2.4.7. Resource Planning, Transportation Functions, Information Management, Factory Simulation, Engineering Function, Tooling and Fixtures Management Entities

Of these entities Resource planning, Factory simulation, Engineering function, Tooling and Fixtures management entities have all yet to be designed and implemented, in either FIX DMACS or in Visual Basic. Resource planning is the entity within the factory controller responsible for scheduling resources to enable jobs to be produced on schedule. This entity must be able to take into account stock on hand, purchases and future orders. Factory simulation is an important function that will allow the operator to simulate the current factory set up inside a virtual factory so as to investigate if it is operating effectively. Once the factory has been simulated and the optimal set up has been obtained, then these settings can be employed in the real system.

The engineering functions of the factory controller are concerned with the design of the work piece for the manufacturing process. It is possible by combining a CAD and CAM package with the factory controller to enable jobs to be designed and then processed, to produce machine readable files. These machine files can be used by CNC devices to automatically process a job.

The transportation and information management entities have as yet to be fully implemented. Some functions of the factory controller so far implemented require

information transportation. However, this function can not be fully completed until all the entities that may require these services have been identified.

## 5.3. Communications

The hybrid nature of the FMS control system means that communication aspects take on special importance. Rather than the traditional centralised control architecture most of the inter-entity communication is achieved through software, the hybrid controller requires a communication medium that allows all entities to interact and transport information as well as broadcast messages to all entities within the factory. Therefore a communications medium capable of transferring job files, machine data files, job descriptions, bidding objects, cell management information and cell data was required. A network was also required that would be compatible with FIX DMACS, a LAN would be best suited for this, preferably of a Bus configuration, such as Ethernet. Ethernet uses a CSMA/CD protocol, which allows a number of communicating devices to effectively transmit simultaneously on the network from point to point. The network currently operating within the department of Production Technology, an Ethernet LAN running IMB LAN software was up to this task. This LAN operates a CSMA/CD data protocol. Further information about the CSMA/CD protocol is available in appendix A.16.

### 5.3.1. FMS Communication

The LAN in the case of the FMS controller is used to provide a communication link between the factory controller and the cell controllers. This link is available to all the cells on the network and one controller will have priority to the network resource. The LAN will enable the auction messaging, IE data, status and control information as well as file transfer between the different controllers upon the network.

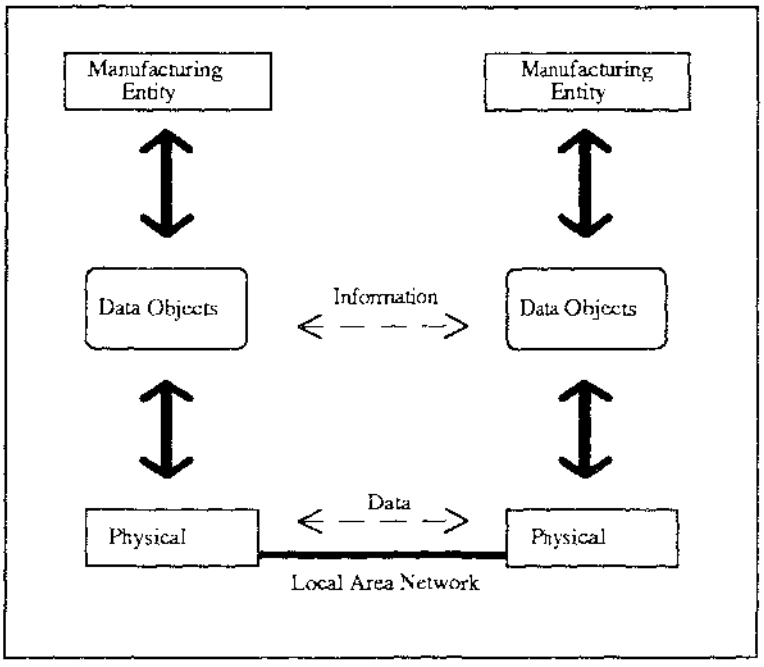



Figure 5.18
Model of IE communications

Figure 5.18 demonstrates how two different entities communicate over the Ethernet LAN. These two entities can be physically separated by short to medium distances, ie the conceivable distances within a medium size manufacturing facility. It was decided early on that whilst MMS/MAP is an excellent communications protocol for the manufacturing industry, it was too complex and the implementation would be out of the reach financial of this research group.

## 5.4. The Flexible Manufacturing System Control Data Model

### 5.4.1. Scope of the Flexible Manufacturing System Control Data Model

The data model is a model that demonstrates the information flow between different parts in the control system. Due to the nature of the control model used for the design of the hybrid FMS control system, many of the elements within the system are only connected to one another by a pre-defined communications channel. In order for one entity or application to communicate to another it must conform to the information transportation requirements of the entity it wishes to communicate with, ie it must have knowledge of the data object group it utilises

Due to the relatively light coupling of the distributed control system the communication aspect of the system is very important. In a centralised system all the entities within the system exist at a single point, however in the hybrid system all the cell controllers are spread throughout a physically distributed area, communicating only via a Local Area Network. The scheduling method employed is communication intensive one, requiring job descriptions and auctioning information all to be transported upon the network.

### 5.4.2. The Data Model

The data model shown in figure 5.19, describes the major components of the FMS control system in terms of the applications used. This model demonstrates the major paths of communication that flows between different platforms with the control system. Information often is transported between entities within the same platform, such as FIX DMACS. The data model describes eight major forms of communication, most that have been defined, and can be described with a group of data objects, others where only the method of communication is defined as the specifics vary from one instance to the next.

All communication between entities created within Visual Basic and FIX DMACS has been defined explicitly by a number of data object groups. Groups of data objects are also used to describe the communications link between entities within FIX DMACS.

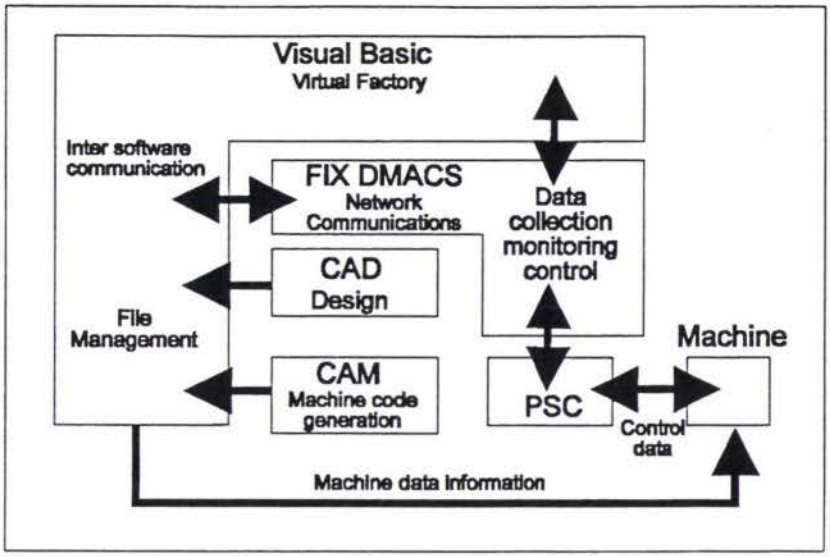



Figure 5.19
FMS Data Model

The following sections detail the structure, where possible, of each of the eight different forms of communication.

These descriptions of information transportation are important as they give descriptions of FIX DMACS database blocks and Visual Basic data structures and the links between them. A detailed description of the type of data flow and where possible a description of its form as a data object is given in appendix A.17.

### 5.4.2.1. File Management

File management is a task of the Visual Basic application that must manage data files from both CAD and CAM packages. The communications link between the front end Visual Basic application and either of the CAD/CAM applications has yet to be formally defined. It is reasonable to say that the communications link will be provided in the form of data files being produced by either of the applications and stored in the appropriate location on the network server.

### 5.4.2.2. Machine Data Information

The machine data information is information that is transferred between the Visual Basic application and the low level device. Once a part has been designed by the CAD application then the appropriate manufacturing plan has been specified by the CAM application and a machine data file has been created, this file must then be transported to the appropriate low level device. The machine data file contains the program that the low level device will use to manufacture the part designed by the CAD application. The DCA instructs the FIX_COM application to transfer the machine data file via an available RS-232 communication port to the target CNC device.

#### 5.4.2.3. PSC - FIX DMACS Communication (Control Data)

The PSC to FIX DMACS communication, is concerned with the transportation of control data or control information from the PSC (low level device controller) to FIX DMACS and visa versa. This transportation of information is handled by the I/O driver. The I/O driver is software that connects FIX DMACS to the low level device controller, or in this case the PSC. The I/O driver allows information of any type to be passed to and from the low level device controller, so long as the information is in an appropriate form for the FIX DMACS database block.

The information flowing across the link can either be control information going from FIX DMACS to the IE, or data information flowing from the IE back to the cell controller. The link between FIX DMACS and the IE can really only be defined in terms of the VMD at the cell controller side of the link, as the actual specifics of the IE have not yet been implemented. Thus the complete listing of the current VMD as it has been so far defined can be found in Appendix B.

#### 5.4.2.4. Inter-Software Communication

Inter-software communication is the communication that occurs between two different applications. In the case of the data model this refers to communication that occurs between any Visual Basic application at either the cell or factory controller and FIX DMACS. Each of the entities participating within the communication must have knowledge of the communication interfaces of the other entities, they wish to communicate with. As many Visual Basic applications are currently functioning within the FMS, the form in which information is transported has been strictly defined, these definitions are called data object groups.

The information transportation as discussed previously can be undertaken in either one of three ways.

1. DDE transfers, this transports information via Dynamic Data Exchange between FIX DMACS and a Visual Basic application.

2. Transfers within a FIX DMACS node or cell, this is achieved by coping data from one block to another.

3. Transfer between FIX DMACS nodes via the view screen, this uses FIXDMACS own programming language to transfer information between two data base blocks. A complete list of all data object groups and data object mapping's can be found in appendix A.17.

## 5.5. Limitations to Implementation

Most of the limitations to the development of the FMS control system implementation have been previously described in chapter 3. These limitations focus mainly upon the lack of resources that have hindered the development of the project. Difficulties with computer resources, network resources and financial resources are all common aliments in this sort of project. One major limitation encountered however that was not anticipated, was the lack of a link between the cell controller and the IE. This development of the link or I/O driver as it is called, proved an relatively difficult task and as yet has not been completed. This placed a cap on the goal of completing a skeleton operating system as, it was not possible to connect the IE to the cell controller. The vital sections of the factory and cell controllers were completed, yet problems in completing the link between the cell controller and the IE have stalled development of the IE itself. Thus whilst some development of the IE has been undertaken a substantial amount of work still needs to be conducted in order to produce a minimal operating system.

## 5.6. Conclusion

In this chapter the implementation of each entity of the FMS control system has been discussed in full. Furthermore the operation of each of these entities has also been discussed. The cell controller was implemented with a combination of FIX DMACS data base blocks and Visual Basic applications and has seen the most substantial development of all three layers of the model. The factory controller still requires substantial research to be completed, so far only the entities that were required to interact with the cell controller have been implemented. The Intelligent Entity has seen the least amount of development work. The lack of an I/O driver has precluded most development work on the IE. Thus in this chapter most of the discussion on the IE has focused on taking the concept of the IE a step closer to the implementation rather than discussing the implementation itself.

The implementation discussed in this chapter details the FMS control system that has been developed to a partial state of completion. The system as it currently exists consists of a partially completed cell controller capable of negotiating for jobs upon the network and then processing these jobs. However, this functionality has yet to be mapped onto the factory floor via a number of IE's and thus is still a distance from completion. The factory controller at present is capable of allowing the user to view basic information about the cell controllers and to allow the user to define jobs and the configuration of the factory. The factory controller is also capable of acting as the decision maker in the auction process.

# CHAPTER 6

## DISCUSSION AND CONCLUSIONS

Table of Contents

## 6.1. Introduction

The research conducted for this thesis has focused on two areas, the implementation of the FMS control system using a SCADA package and the development of an auction based scheduling system in the same environment. The implementation of the FMS control system was taken to the point where the auction algorithm could be implemented and tested in an appropriate frame work. The auction algorithm was evaluated through simulation, for functionality and performance. The following discussion and subsequent conclusions, focus upon these two areas and discuss the suitability of FIX DMACS for the implementation of the FMS control system as well as the operation of the auction algorithm in terms of the experimental evaluation.

## 6.2. The Implementation of the Flexible Manufacturing System Control using SCADA Software

At the conclusion of the this first stage of research, it was felt that FIX DMACS was a capable platform for the implementation of the FMS control system. Specifically, it is capable of being used as a tool for implementing portions of a hybrid control model. FIX DMACS was useful for a number of reasons;

1. It is able to operate in a distributed fashion. Therefore it can provide the network communication backbone for the FMS control system, transporting a variety of information from bidding objects to machine state information.

2. In comparison to other manufacturing control systems it is low cost.

3. Some of the software frame work required for a manufacturing control system has been created, FIX DMACS provides the ability to control low level devices at the factory floor.

4. A consistent user interface. FIX DMACS operates within the Windows environment and thus can be interfaced with a multitude of other Windows applications.

5. It is possible to integrate human operators into the manufacturing control system, using FIX DMACS, user definable operator screens.

The points above highlight why FIX DMACS was suitable for the implementation of the FMS control system. Each of these points are discussed in terms of the implementation undertaken.

## 6.2.1. Experimental Method

In order to investigate the operation of the bidding algorithm within the FMS, a number of experiments were conducted. Firstly the experimental factory was run with test data to prove the operational merit of using FIX DMACS to implement the FMS control system. Once this was successful, five experiments were conducted, the first of these experiments was conducted in order to demonstrate the operational bias of the auction algorithm towards cell controllers with lower cell controller numbers (experiment 1, appendix M).

This bias is consistent and meant that certain cells are favoured over other cells when the result of the auction is tied, ie when two cells returned the same winning bid. Experimentation upon the operating system had shown that this case in medium to low levels of system load was a relatively common occurrence, especially in a under utilised system where often two cells will place a bid of zero. This bias was demonstrated through conducting a simple manual simulation using the same functionality as the bidding algorithm implemented within the FMS control system. A number of tasks $T_1$ to $T_{10}$ were created with a random length and then were release in to the theoretical factory at time T=0. These jobs were distributed amongst the devices which were all of the same type, through the use of the EFT bidding algorithm, as shown in figure 6.1.

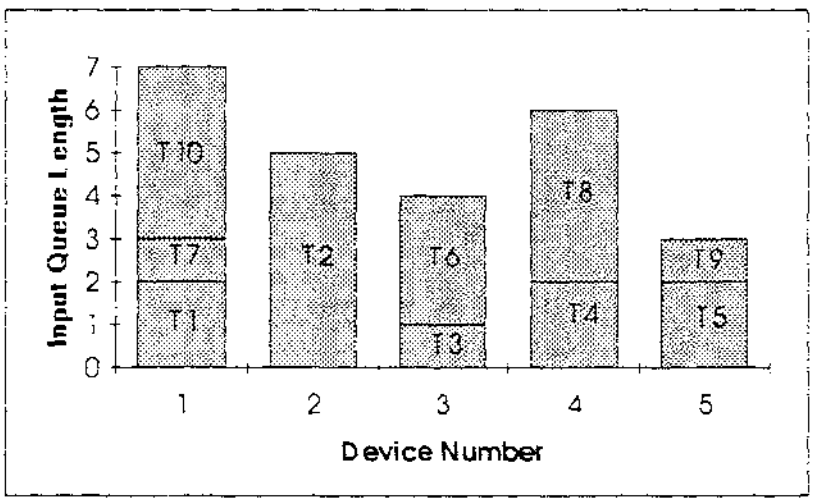



Figure 6.1
Queue loading over five devices

Figure 6.1 shows the bias towards the devices on the left hand side. This is because when the bidding algorithm is making the decision to allocate the job, if two cells/devices have placed an identical bid then the cell/device with a lower number will always win the bid. This bias was further demonstrated in the second and third simulations, (experiment 2, appendix M).

The second experiment was conducted in order to determine the functionality of the auction algorithm. It was important to assess the operation of the algorithm as several jobs were released into the factory (experiment 2, appendix M). This experiment consisted of two simulations both of which were conducted manually. A theoretical factory was created consisting of four cells, each containing either three or four machines, of a different type. The choice of which of the five different types of machines would be placed in each cell, was done randomly. In the case of the first simulation, eleven randomly created jobs (see figure 6.5 for job definition) were released into the system for processing, using the EFT bidding algorithm. All the jobs in this simulation were released at time T=0. In the second simulation a theoretical factory was created which consisted of three cells, each with either three or four machines. Eighteen jobs were released into the factory and were distributed amongst the three cells. In this case the jobs were released into the system in three batches at a randomly selected interval. The results of these simulations demonstrated the functionality of the auction algorithm, as seen in figure 6.2 and 6.3.

Figure 6.2
Job flow diagram of experiment 2, simulation 1

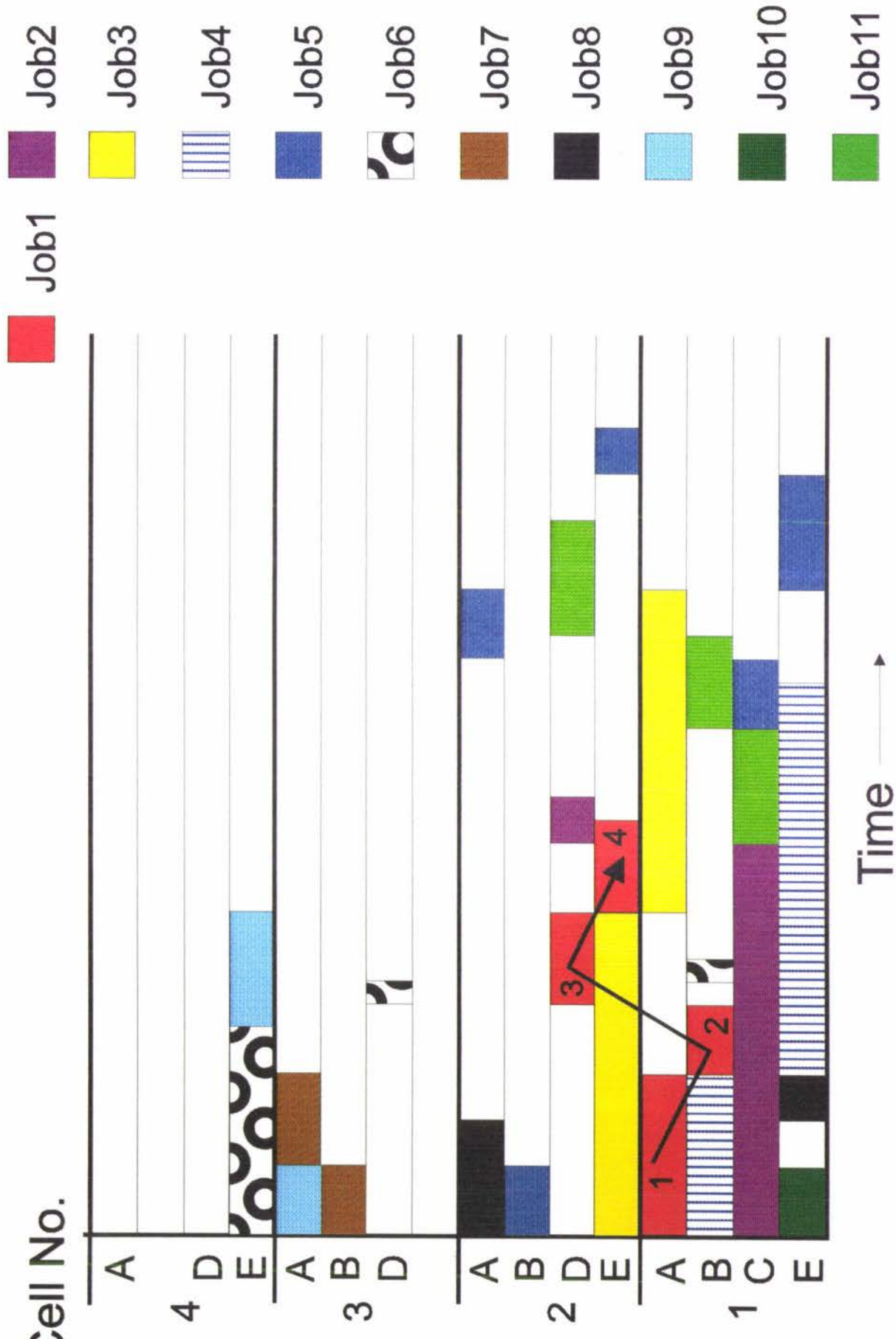


Figure 6.3
Job flow diagram of experiment 2, simulation 2

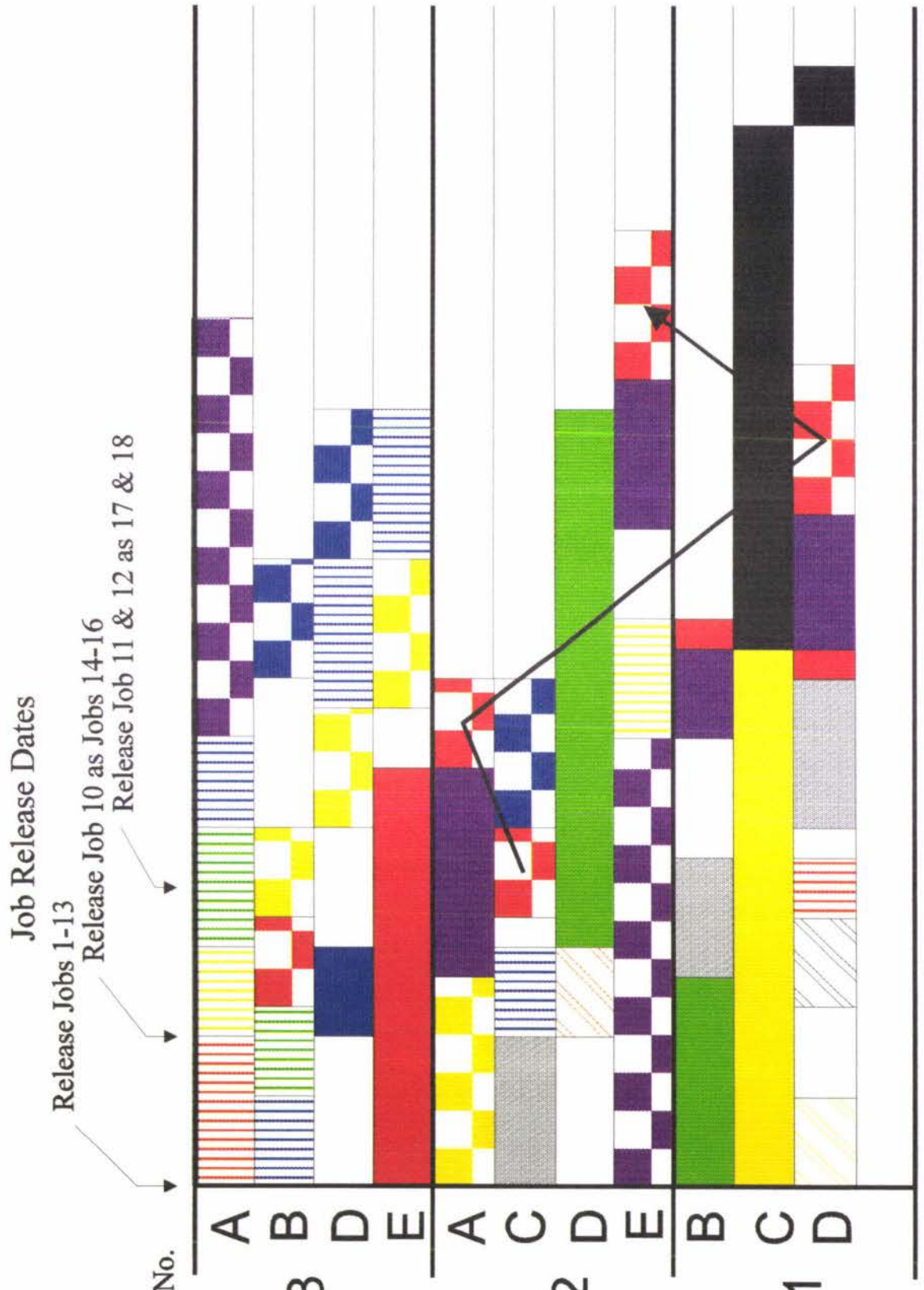


Figure 6.4
Legend of jobs for figure 6.3

Figure 6.5
Definition of jobs used in experiment 2, simulation 1 & 2

The third experiment was conducted in order to evaluate the effectiveness of the Earliest Finishing Time auction algorithm (experiment 3, appendix M). This simulation was conducted with a purpose built simulation package, by a simulation research group (Ginting 1994b) within the Department of Production Technology. This research group utilised the aforementioned EFT bidding algorithm with the aim of creating a comprehensive simulation of the FMS bidding algorithm.

This simulation was undertaken through the construction of a computer based simulation model. This model was created to as closely represent the operation of the auction algorithm within the FMS control system as possible. The model of the auction algorithm used in this simulation, considered a more realistic assessment of EFT. Whilst in the FMS auction algorithm transportation time is assumed to be zero, in this simulation the transportation time was set to a constant. The computer simulation itself was written within Visual Basic. The scheduling of jobs was simulated within a factory consisting of four cells, one cell containing two machines, another containing three machines and the other two containing four machines. Data from this simulation was collected after it had been verified that the model had achieved a steady state. The simulation was then run a number of times whilst in the steady state condition, in order to assess the effectiveness of the EFT auction algorithm.

Experimental data from these experiments is discussed in section 6.4. Further information can be found in appendix M.

Once the implementation of the three layered FMS model had been completed as far as possible, it was then required to be fully tested. To do this a fourth experiment was conducted upon the 'real' system. The aim of this experiment was to test the operation of the partially operating FMS control systems bidding algorithm. The operation of a factory was simulated through implementing three controllers for the FMS control system software upon three separate IBM compatible 486 personal computers, consisting of two cell controllers and one factory controller. These computers were all connected via a local area network, which enabled the distributed auction elements to communicate.

The two cell controllers were identical to one another except for their address upon the network and were configured with two devices on each cell controller either of type A, B or C. A series of simulations were then conducted upon the 'factory'. These experiments were conducted under the same conditions, however the job release dates and the job release order was randomly selected and thus varied between simulations. To conduct the simulation a number of jobs were then released in to the factory and were processed in real time. The processing of the jobs at each virtual manufacturing device was undertaken by a simulation program, which simulated the actions of the device control algorithm and the IE. A detailed discussion of the results and experimental method of this simulation can be found in section 6.4 and in appendix N.

The set up of the hardware for the simulation of the EFT auction algorithm upon the real implementation of the flexible manufacturing control system is shown in figure 6.6.

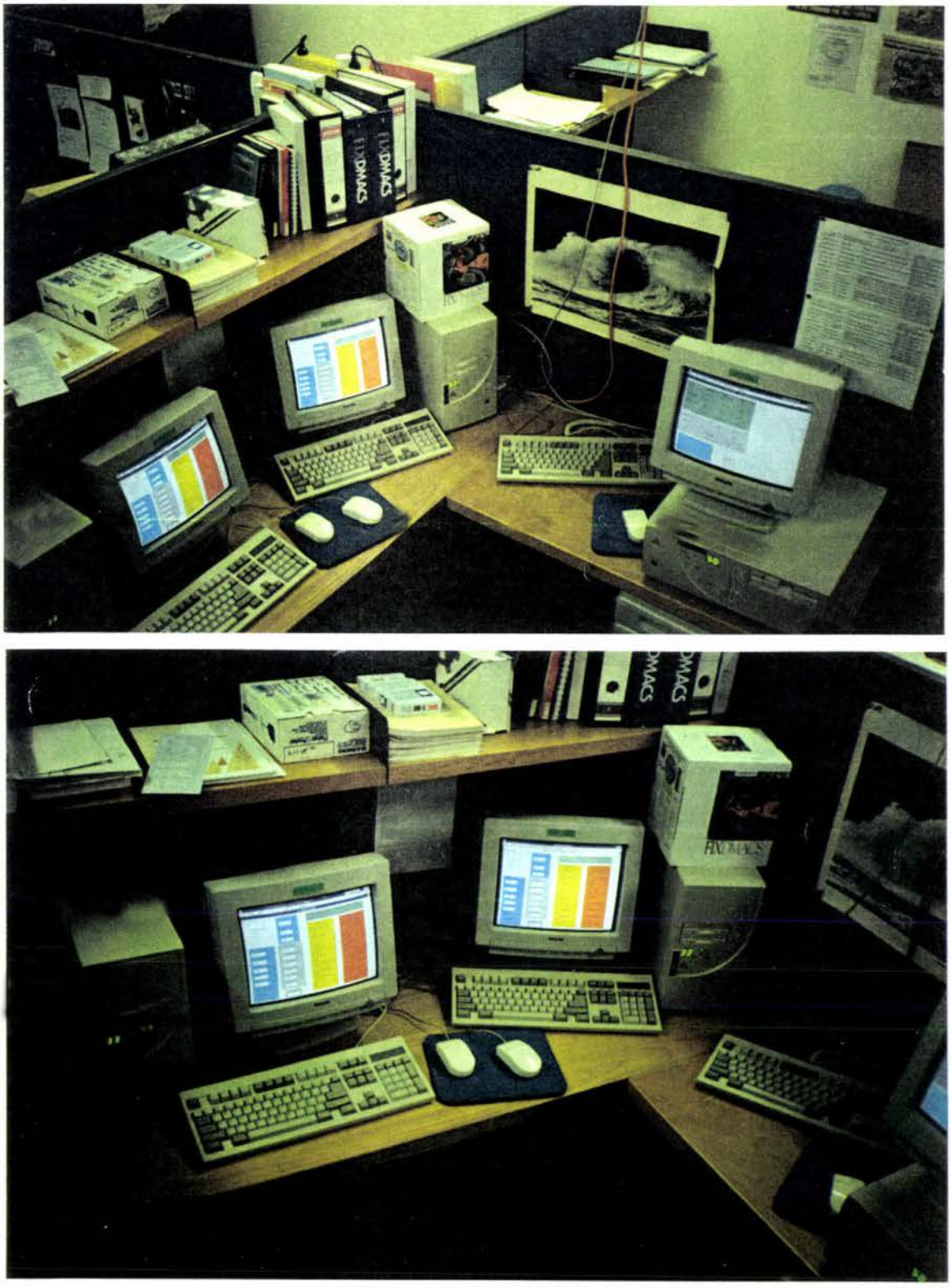


Figure 6.6
Layout of computer hardware for simulation of the EFT auction algorithm

In a final experiment, the 'experimental' factory was evaluated for the effect of database scan times upon the operational speed of the cell controllers. The experimental factory operated at approximately 1000 times slower than its maximum capability. This was done in order to allow the experimenter to follow the operation of the auction process. A experiment was conducted by running multiple simulations of the same bidding operation whilst varying the scan time. This provided information about how the FIX DMACS database was effected by changes in scan time. It also gave an insight in to the maximum possible speed at which the bidding algorithm could operate at. Section 6.4. discusses this experiment further, detailed experimental method and data from this experiment can be found in appendix L.

### 6.2.2. Distributed Database

FIX DMACS ability to operate in a distributed fashion was probably the single most important factor in its usefulness for the implementation of the hybrid FMS control system. This ability made FIX DMACS ideal for providing the manufacturing systems communications backbone. The current implementation has demonstrated (see section 6.4) through using the three layered hybrid model that a number of cells and a factory controller can be implemented using several different FIX DMACS nodes, each on a separate computer. This "factory" can then operate using an opportunistic scheduling method to distribute jobs amongst the cells, with a LAN as the communication medium.

The FMS control system requires a number of "intelligent" cell controllers to co-operate in order to achieve the goals of the system. As each cell controller has a separate FIX DMACS database and has the necessary "intelligence" to;

1. Assess its capability to undertake a particular job being auctioned, and deliver this assessment to the auction decision making entity.

2. Initiate an auction when it has a partially completed job which requires further operations, "tasks" to be performed to complete it.

3. Undertake required operations upon jobs it currently has stored in its incoming queue, using one of several low level devices it controls.

4. Degrade gracefully from faults that occur in the low level device under its control. Jobs that are waiting in the cells incoming queue and can no longer be performed, must be re-distributed amongst the remaining cells.

5. Monitor the progress of each of the low level devices under its control in real time for operations which are not highly time critical.

6. Display and report the current state of the cell at both the local node and also at any other node in the factory if required.

This "intelligence" is located at each cell and distributes the decision making capabilities of the factory as a whole, amongst all the entities within the factory. Rather than formulating a schedule at a centralised processing unit, the components for the schedule are calculated at each cell. Through using the LAN to allow each cell controller to communicate with the factory controller, the cells in the factory can operate for all intensive purposes, completely independently of one another, with each cell controller only aware of its local environment and goals.

The complete operation of the cell controller, so far has not yet been fully tested, a modest 'experimental factory' has been set up consisting of two cell controllers and a factory controller. An experiment has been conducted on this experimental factory, with the processing of jobs being simulated (see appendix N and section 6.4). This experiment (experiment 4) on the real system demonstrated how an auction based scheduling method could be used, through implementing system elements (cells) in a distributed fashion. It is fair to say that from this simulation the distributed nature of the hybrid control method has been expressed in accordance with the three layered FMS control model and that FIX DMACS has been successfully used for this purpose.

Using FIX DMACS to distribute the intelligence of the FMS control system brings certain benefits attributed to heterarchical control architectures. Therefore it is felt that the current implementation of the FMS control system using FIX DMACS has numerous potential advantages over traditional hierarchical control systems. However, whilst these benefits have previously been discussed (Duffie 1991), because the implementation has not yet been completed, most of these benefits can not be imperially demonstrated. Two major benefits that were demonstrated by the successful operation of the experimental factory were; increased system robustness and decreased reliance upon a centralised processing unit.

The robustness of the bidding algorithm was tested through running a simulation on the experimental factory, then causing a machine to fail, this showed how the machine that failed was no longer included within the bidding process. This is demonstrated in figure 6.7. here machine 1 in cell 1 was intentionally shut down at time T=73. The effect of this was that machine 1, cell 1 was no longer included in the bidding process and did not receive jobs J6 and J3, which went to machine 1 in cell 2.

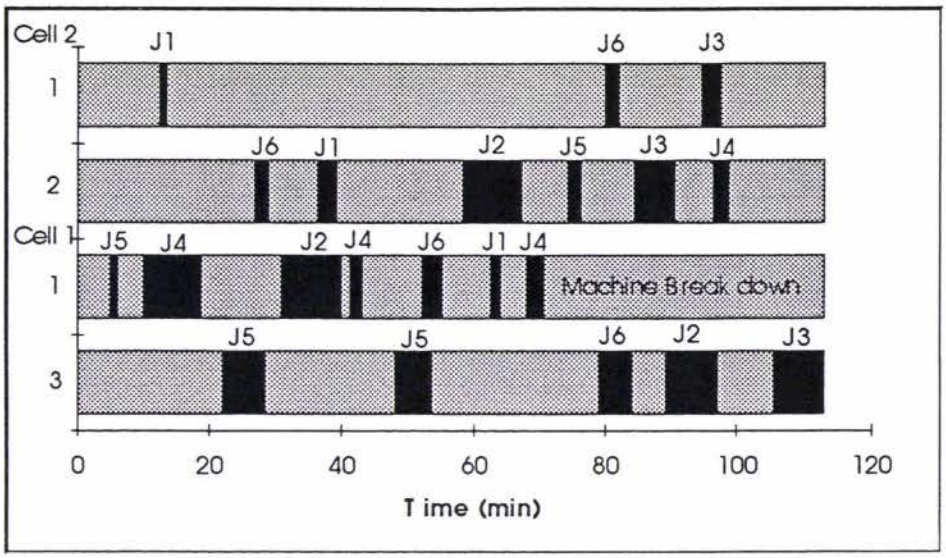



Figure 6.7
Job flow diagram from experiment 4, simulation 2 showing effect of machine failure

Both of these benefits were achieved because the scheduling decision occurs only at the time when a part requires processing. Thus any changes in system status can be taken into account such as machine failure or queue fluctuations. Other benefits of the hybrid control methodology such as real time scheduling capability, improved scheduling performance, ease of re-configuration and so forth can not be demonstrated until the FMS control system is more fully implemented. However, this does not belittle advantages of the novel control methodology chosen.

Using a distributed methodology the computational requirements of each computing element in the factory is reduced. Rather than requiring one substantial centralised computing element, a number of less substantial computers can be used. The FIX DMACS implementation of the FMS control system demonstrates this point exceptionally well. The speed of operation of the SCADA package was found to be a function of the size of the database. This was demonstrated by taking an existing database, checking its scan rate, then adding more database blocks and checking the scan rate again, as shown in figure 6.8.

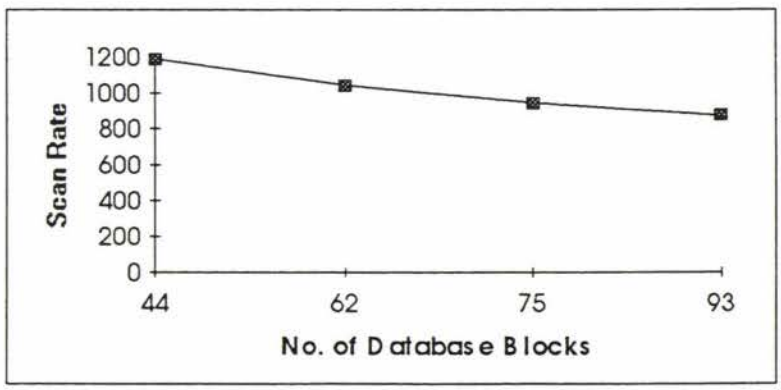



Figure 6.8
Scan Rate (cycles per min) vs Database Size

From figure 6.8 it can be seen that as more database blocks are added the scan rate decreases. Thus FIX DMACS becomes more heavily loaded and less able to interact with its environment in real time. The cell controller database currently has a scan rate of 770 cycles per min. If the entire factory was to be implemented upon a single computer then it is possible that increasing the number of blocks (approximately 1000) could cause the scan rate to drop below 60 cycles per min and the database would become critically loaded.

Therefore it can be said that by distributing the processing requirements of the factory amongst a number of computing elements, it is possible to use low cost computing elements and FIX DMACS to provide real time control of non highly time critical operations. In the case of this FMS control system, the distribution of the processing requirements occurs at two levels, at the cell control level and the IE level. Through this method, the most appropriate computing elements can be used at the most appropriate level. Moderately fast, yet highly intelligent elements at the cell control level and very fast, low intelligence elements at the factory floor, where highly time critical operations are undertaken.

A further advantage of distributing the processing requirements over a number of independent processing elements, is that the FMS control system is inherently resistant to certain types of failure. In the current FMS control system, if any one of the cell controllers develops a fatal fault and shuts down, then so long as it did not have a monopoly on any one type of machine, the factory can recover, and no further jobs will be allocated to that cell. The only type of fatal fault that the system is unable to recover from is a loss of the LAN or a complete shut down of the factory controller.

However, with these advantages came a number of unexpected problems that had to be overcome. These problems occurred when implementing portions the FMS control model within FIX DMACS. The method in which FIX DMACS uses the LAN to communicate between different nodes, is highly suitable for what it was designed for, ie small amounts of timely information. However, in the manufacturing environment often large volumes of information require transportation, such as CNC program files which are often greater than 10Kbytes. The transfer of large amounts of information was unable to be undertaken by FIX DMACS and was achieved instead by utilising a network server as a common storage point for some types of information, whilst this solved the problem it meant that the operation of the FMS control system was now dependent upon a network server for its operation.

A second difficulty encountered was in conforming to one of the essential characteristics of this hybrid controller. Namely that no entity should assume the co-operation of another entity, from section 3.2.1. This characteristic was conceived so as to maximise the independence and the ability of each entity to recover from faults. This meant however, that communications between each cell controller and the factory controller at the software layer were asynchronous, ie no synchronisation between communication entities existed. Whilst making the system robust, it also meant that the communication protocol became quite complicated, as at no time could any entity be guaranteed to receive a desired response to a request.

### 6.2.3. Cost of Implementing the FMS Control System

One major advantage of using FIX DMACS as a basis for the implementation of the FMS control system, was its relatively low cost. In comparison to most other manufacturing control systems, the cost of the SCADA package is only a fraction of the price. (Appendix I includes costing regarding various process control and computer integrated manufacturing systems) The actual development cost of the FMS control system was relatively small, as off the shelf software was used as a basis for the implementation.

The hardware utilised by the FMS control system at this stage was relatively limited, consisting of three IBM compatible 486's (two cell controllers and a factory controller) and a Fisher & Paykel PSC (an IE). Therefore by making use of cheap, off the shelf software and modular hardware, the current implementation was achieved at relatively low cost (approximately $17,000). A further potential advantage that will be achieved as the FMS control system approaches completion, is that the cost of expanding the system will be relatively small compared to a similarly sized hierarchical system (Duffie 1991)

### 6.2.4. FIX DMACS as a Framework for the Implementation of the Control Aspects of the FMS

The control portion of the FMS system is what provides the decision making ability of the factory and cell controllers. This must be capable of reacting to changes in the factory's environment, in accordance to rules which govern the behaviour of the control entity. These control entities were implemented within FIX DMACS and acted as the engines that drove the factory. The rules that governed the behaviour of the control entities were programs that were written in order to give entities the intelligence to make decisions about how to react to changes in their environment. FIX DMACS provided an environment where programs could be written within database blocks, these programs could interact with other blocks in the database, some of which carried state information and were used to effect change in the state of the cell. To enable cell controllers and the factory controller to operate intelligently and in accordance to the characteristics outlined in the design, a large number of programs were required to be written. Many of these programs or control elements operated in parallel to one another, thus different control elements within the factory and cell controllers operated independently of one another.

This method of implementing control within the factory and cell controllers, whilst effective it did introduce several difficulties. There was no master program that contained all the operational logic of any of the controllers. Due to the size limitation of the programmable elements within the database, many small programs were written, each operating in parallel and undertaking a small portion of the control required for a particular controller. This semi parallel programming meant that a number of small independent programs could be substituted for a single larger

algorithm. These small programs where relatively simple to create and operated quickly, however they had the disadvantage of having complex communication requirements.

Whilst FIX DMACS did provide the capability to implement the necessary control required for the factory and cell controllers to operate, it did introduce the problem of implementing these control algorithms in an unstructured semi parallel environment. This environment had the advantage of simplifying the implementation of individual control tasks, however it introduced the problem of co-ordinating the actions of the entire controller.

### 6.2.5. FIX DMACS as a Tool for the Integration of Humans into a Manufacturing Environment

FIX DMACS operates within a Windows environment and thus has all the benefits in terms of user friendliness of a well-conceived user interface. One further advantage of the Micro Soft Windows interface is that all applications within the environment will have a consistent interface. Through using Micro Soft Visual Basic as a programming language it was possible to create user friendly applications in a short period of time that would appear similar in design to all other Windows applications. The user interface to the virtual factory was mostly provided by a Visual Basic application. An application called Virtual Factory was written to enable the user to perform a number of essential functions at the factory layer, such as factory and job definition and monitoring of queue levels. The Virtual Factory application interacted with the user and with the FIX DMACS program, to provide the flexibility necessary to allow users to alter the state of the FIX DMACS application.

A number of operator screens were created within FIX DMACS, these operator screens could be connected directly into any database in the factory. This enables the user to view the current state of any database in the factory. This is a powerful tool that was used in the implementation of the information presentation entities in the FMS control system. Here real time information could be easily displayed to the operator in a variety of different formats. These operator screens also allowed feed back from the user in response to changes in the database. Thus the user was able to be integrated into the manufacturing system as an essential part of the system both at the factory and cell controller layer, through interaction with both Visual Basic applications and FIX DMACS operator screens.

### 6.2.6. Flexibility of the Distributed Architecture Implemented within FIX DMACS

One of the secondary aims of this research was to design and develop a manufacturing control system that maximised the flexibility of the manufacturing elements within the system. Through the design of a "virtual" factory within FIX DMACS it was possible for the user to easily alter the configuration of the factory, this was done by simply altering the virtual software model of the factory to reflect changes in the physical

system. The modular design of the FMS control system meant that to add another cell was a relatively easy operation, a copy of an existing cell is made then altered slightly to give it a new address on the network. The system will then automatically account for the new cell in all operations of the factory. The addition of new devices within the cell is equally as simple, the description of the new device is entered at the factory controller using a library of known devices, then the virtual model of the cell is reconfigured. This re-configuration causes the virtual representation of the device at the cell controller to change and also reconfigures the set up of the IE to take into account the new device. This flexibility is not only beneficial for the purpose of adding new cells and devices, but also in the event of devices or cells being removed due to failure or re-configuration. This flexibility is in fact an inherent characteristic of the three layered hybrid architecture of the FMS control systems design.

The ease of re-configuration and its ability to recover from faults means that the FMS control system has the capability to operate effectively in extremely testing environments. High product variation and machine failure are both situations that can potentially be overcome by the FMS control system. In a test situation the software re-configurability of the FMS implementation has been evaluated, currently a minor software re-configuration (addition of a single machine) will take approximately 40 seconds and a major re-configuration (addition of a new cell) will take approximately 1-2 hours. This only takes into account the re-configuration of software as no hardware has yet been integrated with the control system.

The opportunistic scheduling method employed in the FMS system is inherently resistant to faults. Each physically independent cell in the factory has been designed in such a manner so as no entity in the factory will assume the co-operation of another entity. Thus the failure of any cell will not effect the operation of other cells in any way. Jobs are only scheduled as are required, no assumption is made about the future availability of devices within the factory, as is done with a traditional schedule. Thus if a device or cell fails then that device will simple not be included within the auctioning process and thus will not receive any new jobs to be processed.

## 6.3. Design and Implementation Problems

During the implementation of the FMS control system a number of major difficulties were encountered that altered the scope of the implementation within FIX DMACS. It was first thought that most of the three layered model could be implemented within FIX DMACS, as time progressed it was found that due to the inability of FIX DMACS to store and transfer large amounts of information via the network, only portions of the model could be implemented. Another problem encountered with the implementation was the instability of both the network and the Windows environment, occasionally for no apparent reason a problem would occur with either the LAN or Windows that would cause one portion of the FMS software operating on a cell controller to shut down.

In addition to these difficulties it is also recognised that during the design and development of the FMS control system a number of assumptions were made in order to create a feasible solution. These assumptions effectively make the current implementation unrealistic in comparison to a fully operational system. One such difficulty was the problem of material transportation, because the implementation was strictly software based no interaction with a real material transportation was achievable. Thus there was no need to consider the physical movement of materials around the factory, this problem was temporarily solved through treating the material transportation as an instantaneous event.

## 6.4. Discussion of the Auction Based Scheduling Method

The auction algorithm itself was evaluated both on its own through manual and computer base systems and as it was implemented within the FMS control system. The first group of evaluations were conducted by simulation, a number of assumptions were made about how the auction based system would operate. Such as the part transportation time and the bid calculation time were both set to Zero. Experiments conducted (refer to appendix L) showed the speed of operation of the auction varied considerably as the scan time of the database was altered, this can be seen in figure 6.9.

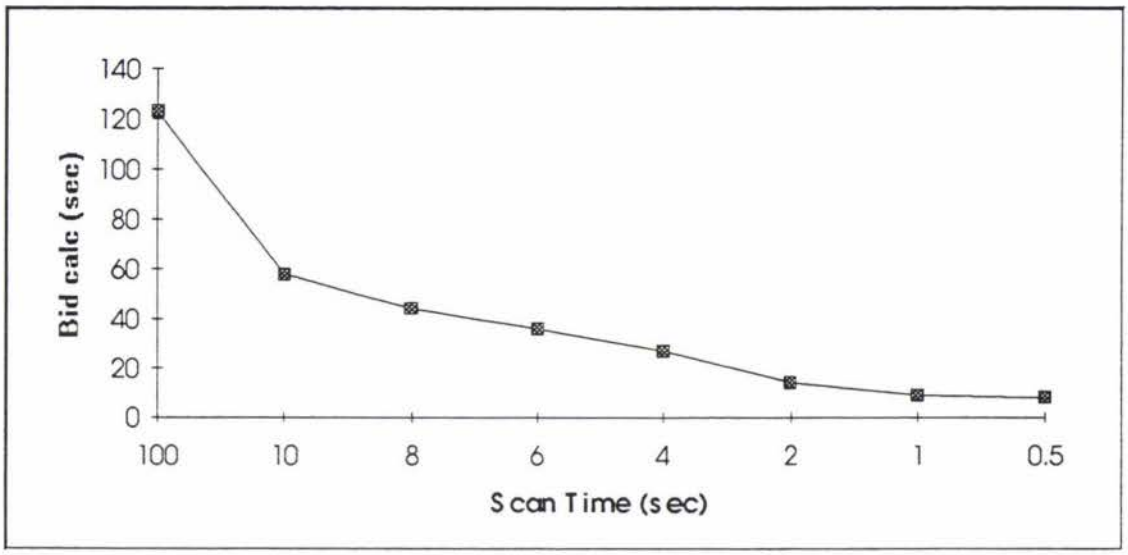



Figure 6.9
Graph of database scan time vs Bid Calculation time for the Auction Algorithm

From this experiment it was decided that when operating, the time taken for the auction $T_1$ (5-15 seconds) to complete, compared with the actual time taken to process a job $T_2$ (greater than 3 minutes) was very different, thus in most cases $T_1 << T_2$ therefore to simplify the simulation, $T_1$ could be negated. Two simulations were conduced where $T_1$ was 0. These simulations were based upon a factory consisting of three cells, two cells having four different machines and one cell having three different machines, with a total of 5 different machine types. The results of the simulations shown in figures 6.2 and 6.3 (experiment 2, appendix M) show how jobs were distributed amongst the machines within the cells.

A further experiment was conducted by the simulation research group within the department of production Technology Massey University (Ginting 1994b). The aim of this research group was to conduct a comprehensive simulation of the EFT bidding algorithm used in the FMS control simulation. This experiment was conducted using a detailed computer simulation model (Ginting 1994a), this simulation accounted not only for process time $T_2$ within the calculation of due data, but also transportation time and network communication time, ie $T_1 \neq 0$. The results of this simulation gave an assessment of how the EFT auction algorithm would perform in a real time

operating system, data from these experiments can be found in table 6.1 and appendix M (Experiment 3).

The results of this simulation were contrasted (table 6.1) with previous experimental data (Shaw 1987) on four different criteria, percentage of late jobs, average lateness, mean flow time and normalised average lateness. However from these results a detailed comparison of the effectiveness of the EFT bidding algorithm used in the FMS control system to other bidding algorithms can not be made. This is due to the different methods in which the two experiments were conducted. It is intended that this simulation program could eventually be used in order to improve the performance of the bidding algorithm in the operational system.

Table 6.1
Evaluation of scheduling methods

| | Shaw 1987 | | | Simulation 3 |
|---|---|---|---|---|
| | Bidding EFT | Bidding SPT | Myopic SPT | Sim Bidding EFT |
| Percentage of jobs late | 20.28 [1] | 33.90 [1] | 36.76 [1] | 56.94 |
| Average lateness (AL) | 3.01 [1] | 3.24 [1] | 4.08 [1] | 34.13 |
| Mean flow time (MFT) | 20.79 [1] | 21.97 [1] | 22.14 [1] | 4.41 |

( [1] These figures are representative of experimental data from Shaw 1987)

A direct quantitative comparison of the data obtained from the simulation done by Shaw 1987 to the data collected from the simulation can not be made as the experimental procedure was similar, yet still had its differences. However, this evaluation is useful in confirming the view that the bidding algorithm performs in approximately the same fashion as previous bidding methods that have been evaluated, as the results obtained were of a similar order that achieved by Shaw 1987.

Whilst this simulation proved through using a similar procedure to Shaw 1987, that the bidding algorithm performed similarly to other bidding algorithms, it did not however evaluate the effectiveness of this algorithm within the partially implemented FMS system. This was achieved through the construction of a small distributed 'experimental factory', which simulated the processing of jobs within each cell. This factory then operated as a number of jobs were released at various points in time, information was collected on how the factory performed.

Due to the small size of the experimental factory set up and because the communication algorithms were operating at approximately 1000 times slower than the potential maximum, the results obtained from this experiment were not comparable to other operating manufacturing systems. No real performance data could be collected from this type of experimentation until the operational speed of the algorithm had been improved to a more realistic level. The experiment was required to operate very slowly and thus $T_1$ was often greater than fifty per cent of $T_2$ and

sometime $T_1 > T_2$. Therefore the aim of this experiment was not to evaluate the performance of the auction based system against other scheduling methods, but was to prove the concept of implementing an auction based system upon the FIX DMACS communications backbone. Figures 6.2, 6.10 and 6.11 demonstrates the machine utilisation and auction communication overheads of an experiment run on the distributed factory. These figures represent the length of time a job spent processing on a particular machine within a cell compared to the total processing time available. The flow of the jobs as it moved between the different machines within the factory can also be scene. Two cells are shown as cell 1 and 2, with cell 1 having two machines of type A and C, with cell 2 having machines A and B.

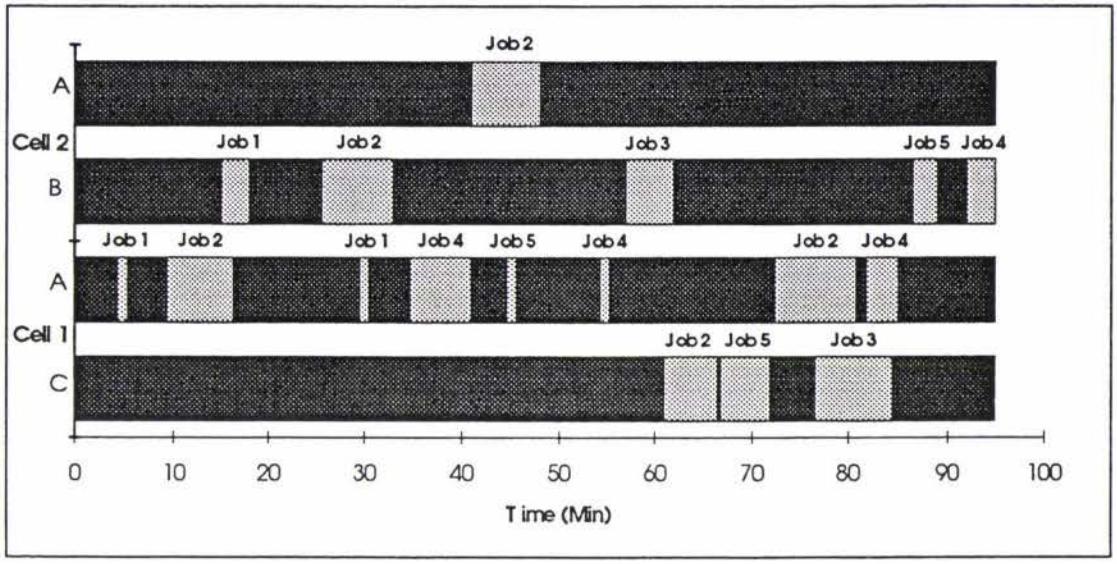



Figure 6.10
Job flow diagram from experiment 4, simulation 1 of experimental factory

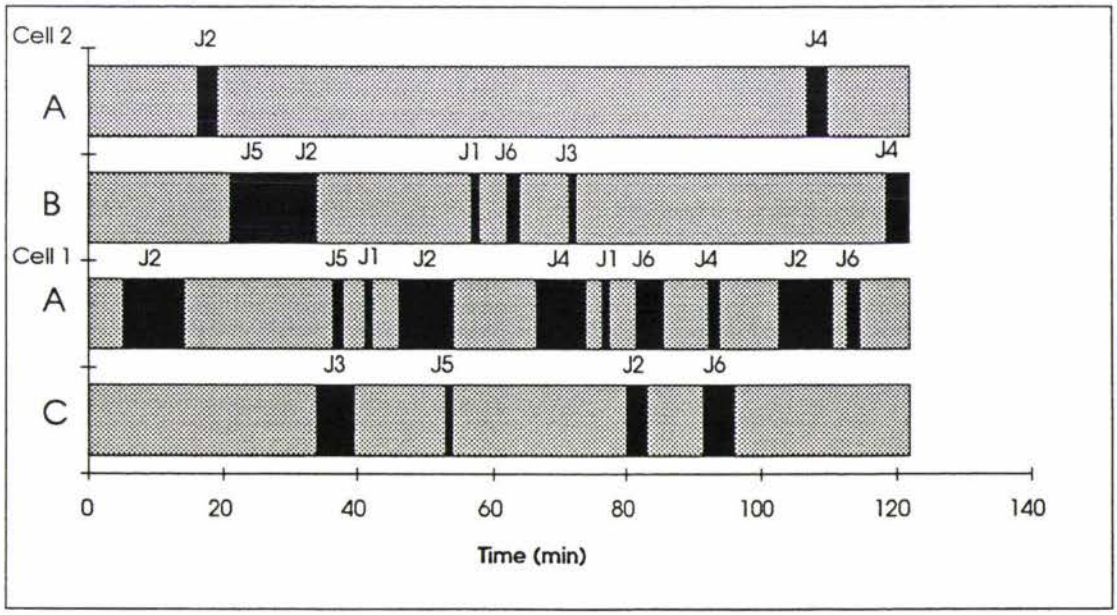



Figure 6.11
Job flow diagram from experiment 4, simulation 3 of experimental factory

From figure 6.10 and 6.11 it can be seen that the production capacity of both cells 1 & 2 are under utilised, this is due to the relative size of $T_1$ compared to $T_2$. It is anticipated that the auction process will, in reality take only a matter of a few seconds ($T_1$ approaching 0 as in table 6.2) for any sized factory (the auction time $T_1$ has been found to be independent of the number of cells in the factory) and thus a more realistic situation can be achieved.

The experimental factory has demonstrated the concept that an auction based scheduling algorithm can be successfully implemented within a low cost SCADA environment. The experimental factory successfully scheduled a number jobs each consisting of up to five tasks and did so amongst four machines from two different cells, each in a physically separate location. The experimental factory ran automatically once all the jobs had been released in to the system from the factory controller. From that point cells were able to simultaneous process jobs in both machines at once and also take part in an auction via the network. Whilst this factory did not operate at high speed, it did however exercise the network communication and bidding abilities of each of the two cells and the factory controller. Therefore it was felt at this stage that an EFT auction based algorithm is suitable for the scheduling of jobs amongst the hybrid FMS control system.

There are however a number of potential problems with the bidding algorithm that could inhibit its performance in the future. The over all goal of each cell according to the bidding algorithm is to obtain as many jobs as possible that can be performed upon the machines in that cell. This is the key local goal that drives each cell. Unfortunately whilst a cell is attempting to achieve this local goal it may be doing so to the detriment of the overall system. This problem becomes apparent when considering the constraint within a factory. No attempt is made by the system to take into account the system constraint, as no cell in the factory is in fact aware of the existence of any other cells in the system. Therefore it can be seen that a sacrifice of efficiency has been made for speed in the case of this simple implementation of the bidding algorithm, when compared to traditional look ahead schedulars. So whilst the auction based system has the potential to approach real time with its ability to schedule jobs, this may not be done in the most efficient manner.

## 6.5. Conclusions

The implementation of the FMS control system in a SCADA package and the development of an auction scheduling system are the two major areas of research which have been focused upon. The implementation of the hybrid FMS control system has been completed to the point where the auction algorithm has been implemented and tested in an appropriate framework. This entailed the, development of a hybrid control model consisting of the factory, cell and intelligent entity layers. Key portions of both the factory and cell controller were then implemented so as to create a minimal system. For the purpose of investigating the auction algorithm, the IE layer has been simulated.

It is felt that this research has succeeded in providing the framework in order to demonstrate the possibility of implementing a FMS control system upon a low cost SCADA package, using low cost software and computing elements. The ability of the auction-based approach to operate successfully within this system, has also been demonstrated through simulation. The achievement of these two goals is a successful precursor for the future development of hybrid flexible manufacturing systems within the Department of Production Technology, Massey University.

In achieving the goals mentioned above a number of novel design concepts have been utilised. The most important of these being the use of many low cost elements to construct a flexible co-operative manufacturing system, which is capable of operating in a physically distributed area via a Local Area Network and is inherently adaptable and resistant to failure. This novel design concept has been imbued within the three layered control model, which differentiates functionally, between three separate layers.

The research conducted in the Department of Production Technology into hybrid flexible manufacturing systems has a different focus to many other similar research groups around the world (Shaw 1987, Tilley and Williams 1992, Veeramani et al 1993). The control architecture employed is a combination of both hierarchical (distributed processing during factory and auction operation) and heterarchical (factory controller provides some global information to allow more intelligent decisions to be made). Furthermore this research is focused upon a low cost, commercially viable solution.

A number of areas of further research have been highlighted; such as the completion of the FMS implementation, constraint identification and the expansion of the role of the factory controller in the auction decision making process, all of which have the potential to greatly improve the performance of this FMS system.

# CHAPTER 7

## FUTURE WORK

Table of Contents

## 7.1 Introduction

During the course of the research associated with this thesis, several areas have been identified that either require completion or would benefit from improvements in the design and implementation. These areas of possible future work have been split up into three different areas. The completion of the current implementation in accordance with the three layered model, the improvement of the three layered model's implementation and the improvement of the auction system. These three areas should combine to complete and enhance the performance of the FMS system. These suggestions for future research stem from the view that future research conducted upon this system should firstly focus upon the completion of the currently implementation before major system improvements are undertaken.

## 7.2 Completion of Current Implementation

The next major goal of this research project should be to focus on completion of the current design and implementation. The three layered model at the factory controller and the Intelligent entity level's still requires detailed completion. Once the finer details of the model have been completed the entire model should be implemented and its functionality tested. It is anticipated that the first step in the completion of the implementation of the FMS model should focus upon the factory controller. This would entail the completion of all the entities specified in the design phase that are not yet completed. These being the factory simulation, master scheduler, fixtures and tools management, manufacturing engineering ie CAD/CAM integration and also the completion of the information presentation phase of the factory management entity.

Once this has been done the IE will next be required to be completed, this is a significant task and can not be started until the connection between the Fisher & Paykel PSC and FIX DMACS has been finished, this is still several months away. The IE would be best implemented at one layer at a time, starting from the top. As the implementation progress downward, the operation of each level can be tested through the use of a number of software test beds. The final stage of the IE implementation should be the connection of the PSC to a low level device, such as the CNC Lathe or Mill, such as is available in the Department of Production Technology.

At this stage an implementation will have been achieved which will allow a full simulation of most aspects of the FMS in operation. However, up to this point a number of assumptions have been made which make the implementation unrealistic. The most significant of these is the lack of material handling and transportation systems. At present materials, parts and tools are assumed to be transported and handled automatically and instantaneously. Obviously this is an unrealistic assumption to make in an operational system. Provision for a transportation system has not yet been made in the three layered FMS control model. However, significant thought has been given for is potential implementation.

It is anticipated that the handling of materials within a cell will be performed by either a human operator or a automatic device of some sort. Because the cell controller will be required to instruct the material handler as to what steps must be taken, this must be an integral part of the cell controller. Therefore it is suggested that a generic virtual material handling device (VMHD) is constructed in each cell controller. The control algorithms in each cell controller can interface with this VMHD to instruct it to perform material handling operations. The VMHD would operate along similar lines to the VMDs within a cell controller. Through this method either a human operator via a computer terminal or a robotic device can be connected to this VMHD. The VMHD will then instruct either of these two devices what material handling operations are required within the cell. If a robotic device is connected to the VMHD, it will be instructed to perform various materials handling operations within the cell, if a computer terminal is present then a human operator will be instructed as how and when to perform these materials handling operations.

The transportation of tools, work pieces and parts between various physical positions in the factory is a complicated task which requires the co-ordination of many cells. This is further complicated by the fact that the scheduling system used is opportunistic, the flow of materials and other objects requiring transportation is not known until the actual transportation resources are required. For this reason it is felt that a similar approach to the scheduling of jobs should be taken to the co-ordination of the transportation resources. It is anticipated that a cell when bidding for a job will also hold an auction to obtain a transportation device, the winning transportation device will then place a reservation into its incoming work queue. The bid placed by the winning transportation device will be a value proportional to the earliest time it can undertake the transportation task, this value will then be included by the cell when placing its bid for a job, thereby giving the cell a more realist assessment of EFT. If the cell does not win the job then it cancels the transportation reservation other wise the reservation stands.

## 7.3 Improvements in the Hybrid FMS Model and Implementation

Throughout the implementation of the FMS model a number of areas have been highlighted in the current implementation that operate sufficiently, yet could be improved. These improvements may form the basis of future research undertaken beyond the completion of the current implementation and fall into two broad categories, improvements in the FIX DMACS and Visual Basic applications.

The Visual Basic applications at present whilst operating satisfactorily are far from optimised. One of the first improvements that could be made in this area is the conversion of all the DDE links to EDA links. EDA links in most cases are a far quicker and more efficient method of transporting data into and out of FIX DMACS database's. This conversion would require the alteration of around four applications, Virtual Factory, Next Queue, Bid Rotation and Com PSC.

A second more adventurous project would be to integrate all of the nine or so Visual Basic applications required at each cell controller into one single application. This would reduce the load on Windows and would increase the stability of the cell controller.

The integration of the CAD and CAM package, is one area where vast improvement can be made. Currently machine files are assumed to have come from a separate CAM package and are used in the factory as such. However, a far easer method of utilising the CAD/CAM facilities would be to integrate them into the Virtual Factory application so they could be easily accessed from within the virtual factory application, the resulting design and machine files would be automatically stored in the appropriate place.

During the course of this research the Visual Basic programming language has been treated solely as a prototyping tool, enabling applications to be easily and quickly created in the Windows environment. Once all the Visual Basic applications had been optimised and were in working order, it would then be useful to re-implement these applications using Micro Soft C++. This would further increase the speed and stability of the applications within the Windows environment.

The second major area where improvements could be made is with FIX DMACS. The implementation of both the cell and factory controllers has partially been undertaken using FIX DMACS. Several parts of this implementation have been identified as being under optimised. One improvement that could be made with the overall implementation of both the cell and factory controllers in FIX DMACS, is decreasing of the SCAN cycle of the database down to the minimum practicable value. Thereby speeding up the operation of FIX DMACS and allowing it to operate in near real time.

The information presentation entities of both the factory and cell controllers are both entities that could benefit greatly from further research. At present the information that is displayed is simplistic and not timely. Research needs to be conducted into the requirements of human operators requiring to integrate with the system. Finally the operation of the cell controller require improvement, various algorithms such as the DCA are presently very simplistic in their operation. These algorithms will require improvements in their functionality to make them able to operate in a more realistic fashion.

## 7.4 Improvements in the Bidding Algorithm

Research conducted into the bidding algorithm in relation to any other part of the FMS control system was probably the most advanced. Therefore it is in this area where the most room for improvement has been identified. At present the bidding algorithm is at a very rudimentary stage, further research will be required to improve the performance this scheduling process to make it more competitive against traditional methods.

Before conducting research aimed at improving the robustness and performance of the auction system, it is suggested that some time is spent in investigating various different implementations on the auction based system. The current implementation uses a bidding EFT approach, however there are many different approaches that can be used. Whilst previous research (Shaw 1987) has shown that this is the most effective algorithm to use, confirmation of this would prove beyond doubt the worth of this approach.

The first and most obvious area that could be improved, is to increase the resilience of the auction algorithm. Currently if a cell has any kind of fault that will cause it to stop processing jobs in any one of its machines, then all the jobs waiting in that machine's queue will stall until the machine resumes operating. A more sensible approach would be to alter the algorithm so if a machine stops for any reason, it will hold auctions to release all the jobs stalled in its queue back into the factory so they can be undertaken by another cell. This improvement has currently only been partially implemented and is not yet functioning. However, when it is, it will demonstrate one of the great advantages of an auction base hybrid system, its ability to adapt to unexpected change.

Further improvements in the auction algorithm can be made, which may not only improve the systems robustness but also improve the efficiency of the overall system. These improvements listed below, are along similar lines and use the same basis for improvement, "that the effectiveness of the bidding algorithm is dependant upon the quality of information, and how that information is utilised". Most of the major benefits from these improvements will come from the stronger expression of the hierarchical aspects of the hybrid control model. Currently the focus of the auction algorithm is with distributed processing. However by placing more emphasis upon the role of the factory controller it is hoped that some of the benefits associated with hierarchical control can be realised.

1. All Information that will impact upon the ability of a cell to undertake a particular task should be taken into account when formulating the bid. The algorithm should test to see if the correct tools are available in the cell, if this is not so then it must arrange transportation for the tools. This in turn may have a time penalty associated with it and therefore should be included in the bidding process. Not only should any time penalties associated with the transportation of tools be taken into account, but also any time penalties associated with the transportation of resources required by a cell to undertake a particular job. These time penalties associated with manufacturing resources will effect the time frame in which a cell can undertake a particular job, in

some cases these resources may not be available and therefore the cell will be required not to place a bid.

2. Currently the machines within a cell operate a batch system, where all the parts in a particular task must be completed before the jobs is handed on to the next cell. The performance of the system could be improved through decreasing the batch size required before the job can be passed on. The minimum sub job size feasible would depend upon several factors which can be grouped into a cost associated to the bidding of a sub job. This cost must be compared to the benefits of decreasing the sub job size. Through this method an optimum sub job size could be arrived at which would maximise the resources of the factory.

3. The performance of the bidding algorithm can be improved not only through considering more information but also through using the available information in more intelligent ways (Chandra et al, 1991, Duffie et al, 1994, Veeramani et al, 1993). One factor that limits the performance of the bidding algorithm is its lack of ability to identify the system constraint. By recognising the system constraint it should be possible for the algorithm to increase the throughput of the system by maximising the utilisation of the constraint. Unfortunately identification of the constraint is no simple task, it is suggested that one of two methods could be utilised. Either imperical-simulation based or a qualitative-rule base approach.

The use of an imperial method would enable a simulation (Shaw 1994, Kim 1994) to be conducted using known jobs waiting to be processed and detailed information about the state of the factory, the resulting information could then be used to re-order the jobs waiting, with the aim of maximising the constraint. The second method of identifying the constraint would operate in real time and use rules that investigate the current state of the system. When a cell completes the criteria associated with these rules it would then be labelled the constraint, the factory controller could then take this into account when allocating jobs amongst cells in the factory.

4. A further improvement in the bidding algorithm could be to consider not only the availability of tools, but also the current set up of the machines in a cell. By considering the current set up of the machines within a cell, the set up time portion of the bid being placed could be adjusted. If no set up is required then the cell would be able to place a more competitive bid than if the set up was required. This effectively gives cells with a particular set up a competitive advantage over other cells when considering certain job families that require similar set ups.

When considering the factory auction system as a whole it can be seen that there are two levels of goals associated with the effective operation of the factory, these are local and global goals. It has been suggested (Shaw 1994) that these two different types of goals can some times be contradictory. In some cases a global goal of "system optimisation" may be contradicted by a cell's local goal, "win as many jobs as possible". The present the auction system operates with only primitive local goals and thus whilst attempting to maximise these local goals, may be doing so to the detriment of overall system performance. Thus the need is seen for an auction based FMS system to consider both local goals and global ones, some of the above points illustrate possible methods of evaluating global goals.

However, it is recognised that whilst the evaluation of global goals such as constraint identification and minimisation of late jobs could improve system performance, it will however increase the computational requirements of the overall system. Therefore in the sprit of a hybrid methodology it is suggested that rather than centralising the evaluation of global and local goals, this should be distributed. By allowing the cell controllers to conduct as much of this evaluation (Duffie 1994) as possible, the computational requirements can be spread. Furthermore a compromise should be made between the benefits associated with added intelligence in the auction decision process, compared to system performance loss due to increased computational requirements.

# REFERENCES


Albus, S. J., Barbera, A. J. and Nagel, R. N., Theory and practice of hierarchical control, National Bureau of Standards, Washington, DC 20234, 1988

Bakker, H., DFMS: A new control structure for FMS, North-Holland Computers in Industry, vol 10, part 1, pp 1-9, 1988

Brill, M. and Gramm, U., MMS: MAP application services for the manufacturing industry, Computer Networks and ISDN Systems, pp 357-380, 1991

Butler, M. D. and Nahavandi, S., Application of a SCADA package for the control of a flexible manufacturing system, Proceedings of The Inaugural New Zealand Postgraduate Conference for Engineering and Technology Students, pp 23-30, 1994

Caven, J. and Jackman, J., An icon-based approach to system control development, IEEE Transactions on Industrial Electronics, vol 37, no 3, pp 259-264, 1990

Chan, D. and Bedworth, D. D., Design of a scheduling system for flexible manufacturing cells, International Journal of Production Research, vol 28, no 11, pp 2037-2049, 1990

Chandra, J. and Talavage, J., Intelligent dispatching for flexible manufacturing, International Journal of Production Research, vol 29, no 11, pp 2259-2278, 1991

CIM Reference Model Committee, Purdue University, A reference model for computer integrated manufacturing from the viewpoint of industrial automation, International Journal of Computer Integrated Manufacturing, vol 2, no 2, pp 114-127, 1989

Control Engineering, PC hardware and software guide, Control Engineering, pp 26-29, Feb 1990

Dackroury, Y. and Elloy, J. P., A proposal for a distributed real time operating system supporting the client/server concept, Distributed Computer Systems, pp 63-69, 1988

Davis, R. and Smith, R., Negotiation as a metaphor for distributed problem solving, Artificial Intelligence, vol 20, pp 63-109, 1983

Duffie, N. A., Non-hierarchical cell control, Manufacturing Cells: Control, Programming and integration, Eds Williams, D. J., Rogers, P., Linacre House, Jordan Hill, Oxford, pp 143-172, 1991

Duffie, N. A. and Prabhu, V. V., Real-time distributed scheduling of heterarchical manufacturing systems, Journal of Manufacturing Systems, vol 13, no 2, pp 94-107, 1994

Duffie N. and Piper R., Non-Hierarchical control of a flexible manufacturing cell, Proceedings of International Conference on Intelligent Manufacturing Systems, Budapest, 1986

Fisher & Pykel, Hardware Manual, Monitor Manual, Module Specifications, Programming Manual, PSC Programmable State Controller, 1993

Ginting, D., Hurley, S. E. and Nahavandi, S., Development of a simulation module for near real-time scheduling of process activities in a flexible manufacturing system, Proceedings of The Inaugural New Zealand Postgraduate Conference for Engineering and Technology Students, pp 23-30, 1994a

Ginting, D., Development of a simulation module for near real-time scheduling of process activities in a flexible manufacturing system, Report from Research in Flexible Manufacturing Real Time Scheduling Systems, Department of Production Technology, Massey University, 1994b

Golenko-Ginzburg, D. and Sinuary-Stern, Z., A multilevel production control model for FMS with disturbances, IEEE, pp 81-84, 1988

Graefe, U. and Thomson, V., A reference model for production control, International Journal of Computer Integrated Manufacturing, vol 2, no 2, pp 86-93, 1989

Gupta, Y. P. and Goyal, S., Flexibility trade-offs in random flexible manufacturing systems: A simulation study, International Journal of Production Research, vol 30, no 3, pp 527-557, 1992

Hatvany, J., Intelligence and cooperation in heterarchical manufacturing systems, Robotics, CIM, vol 2, pp 101-104, 1985

Intellution, FIXDMACS software manual, vols 1 & 2, 1993

Intellution, FIXDMACS users guide, 1993

Jones, A., Barkmeyer, E., Davis, W., Issues in the design and implementation of a system architecture for computer integrated manufacturing, International Journal of Computer Integrated Manufacturing: Special Issue on CIM Architecture, 2(2), pp 65-76, 1989

Jones, A. and Saleh A., A multi-level/multi-layer architecture for intelligent shopfloor control, International Journal of Computer Integrated Manufacturing, vol 3, no 1, pp 60-70, 1990

Kim, H. M. and Kim, D. Y., Simulation-based real-time scheduling in a flexible manufacturing system, Journal of Manufacturing Systems, vol 13, no 2, pp 85-93, 1994

Kochhar, C. and Morris, R. J. T., Heuristic methods for flexible flow line scheduling, Journal of Manufacturing Systems, vol 6, no 4, pp 299-313, 1987

Kohler, C. and Schmier, K., CIM Technological and Organisational Change in West Germany, Journal of Manufacturing Systems, vol 10, no1, pp 21-31, 1991

Kohler, C. and Schmierl, K., Computer-integrated manufacturing (CIM)-technological and organisational change in the West German capital goods industry, Journal of Manufacturing, Systems, vol 10, no 1, pp 21-32, 1991

Lewis, W., Barash, M. M. and Solberg, J. J., Computer integrated manufacturing system control: data flow approach, Journal of Manufacturing Systems, vol 6, no 3, pp 177-191, 1987

Lewis, W. C. Jr., Data flow architectures for distributed control of computer operated manufacturing systems: structure and simulated applications, PhD dissertations, School of Industrial Engineering, Purdue University, May 1981

Mahmood, F., Dooley, K. J. and Starr, P. J., An investigation of dynamic group heuristic's in a job shop manufacturing cell, International Journal of Production Research, vol 28, no 9, pp 1695-1711, 1990

Maimon, O. Z. and Gershwin, S. G., Dynamic scheduling and routing for flexible manufacturing systems that have had unreliable machines, Operations Research, vol 36, no 2, pp 279-292, 1988

Malone, T. W., Fikes, R. E. and Howard, M. T., Enterprise: a market-like task schedular for distributed computing environments, Working Paper, Xerox Palo Alto Research centre, USA, 1983

Mandelbaum, M., Flexibility in decision making: an exploration and unification. Ph.D. Thesis, Department of Industrial Engineering, University of Toronto, Canada, 1978

Mandelbaum, M. and Brill, P. H., Examples of measurement of flexibility and adaptively in manufacturing systems. Journal of Operations Research Society, vol 40, no 6, pp 603-609, 1989

Micro Soft, Visual Basic Professional Edition Software Manuals, vol 1-4, 1993

Nagarur, N., Some performance measures of flexible manufacturing systems, International Journal of Production Research, vol 30, no 4, pp 799-809, 1992

Nahavandi, S. and Preece, C., A low cost system integration tool for FMS, Proceedings of 2nd International Conference on Computer Integrated Manufacturing, vol 3, pp 280-285, 1993

O'Grady P. J. and Lee, K. H., An intelligent cell control system for automated manufacturing, Programming and integration, Eds Williams, D. J., Rogers, P., Linacre House, Jordan Hill, Oxford, pp 151-173 , 1991

Perkins, J. R. and Kumar, P. R., Stable, distributed, real time scheduling of flexible manufacturing/assembly/disassembly systems, IEEE Transactions on Automatic Control, vol 34, no 2, pp 139-148, 1989

Ramamritham, K. and Stankovic, J., Dynamic task scheduling in hard real-time distributed systems, Proceedings of Distributed Computing Systems, IEEE Computer Society Press, 1984

Rogers, P., Representation of the cell control task, Manufacturing Cells: Control, Programming and integration, Eds Williams, D. J., Rogers, P., Linacre House, Jordan Hill, Oxford, pp 10-36, 1991

Shaw, M., A distributed knowledge based approach for intelligent manufacturing information systems, Ph.D. Dissertations, Purdue University, U.S.A., 1984

Shaw, M. J., A distributed scheduling method for computer integrated manufacturing: the use of local area networks in cellular systems, International Journal of Production Research, vol 25, no 9, 1285-1303, 1987

Shaw, M. J. P., Distributed planning in cellular flexible manufacturing systems, INFOR, vol 25, no 1, pp 13-25, 1986

Smith, R. G., The contract net protocol: high-level communication and control in a distributed problem solver, IEEE Transactions on Computers, vol 29, pp 1104-1113, 1980

Tanenbaum, A. S.A, Computer Networks, Prentice-Hall International Editions, 2nd ed, 1989

Tilley, K. and Williams D. J., Modelling of communications and control in an auction-based manufacturing control system, Proceedings of the 1992 IEEE International Conference on Robotics and Automation, pp 962-967, 1992

Veeramani, D., Bhargava, B. and Brarsh, M. M., Information system architecture for heterarchical control of large FMSs, Computer Integrated Manufacturing Systems, vol 6, no 2, pp 76-92, 1993

Weston, R. H., Gascoigne, J. D. and Sumpter, C. M., Intelligent Interfaces for Robots, IEEE Proceedings, vol 132, no 4, 1985

Weston, R. H., Harrison, R., Booth, A. H. and Moore, P. R., A new concept in machine control, Computer Integrated Manufacturing Systems, vol 2, no 2, 1989

Weston, R. H., Gascoigne, J. D., Sumpter, C. M. and Hodgson A., Robot integration within computer integrated manufacture, International Journal of Production Research, vol 27, no 3, pp 515-528, 1989

Weston, R. H., Coutts, I., Hodgson, A., Murgtroyd, S. and Gascoigne, J. D., Generally applicable cell controls and examples of their use, Manufacturing Cells: Control, Programming and integration, Eds Williams, D. J., Rogers, P., Linacre House, Jordan Hill, Oxford, pp 10-36, 1991

Weston, R. H., Gascoine, J. D., Rui, A., Hodgson, A., Sumpter, M. C. and Couttis, I., Steps towards information integration in manufacturing, International Journal of Computer Integrated Manufacturing, vol 1, no 3, pp 140-153, 1990

Williams, T. J., One view of the future of industrial control, Control Engineering Practice, vol 1, no 3, pp 423-433, 1993

For the children, I'll never forget.

This love that heals like a crooked limb
In each of us, source of our grief,
Could tell us if we cared to listen, why
Sons by mayhem, daughters by harlotry
Pluck down from the sky's rage on settled houses:
The thin girl and the cornerboy
Whose anger masks their love
Unwind, unwind the bandages
That hide in each the hope of joy.

For me it is the weirs that mention
The love that we destroy
By long evasion, politics and art,
and speech that is a kind of contraception:
A streetlight flashing down
On muscled water, bodies in the shade,
Tears on a moonwhite face, the voice
Of time from the grave of water speaking to
Those who are lucky to be sad.

(1963; Letters from Pig Island, CP 281)
James K. Baxter

# APPENDICES

## Table of Appendices

# Appendix A

## Supplementary Information for Chapters 2,3,4,5 & 6

This appendix contains information in addition to the discussion within sections of chapters 2, 3, 4, 5 and 6. This additional information provides a more detailed description of the respective topic.

# Appendix 1.1

4.3.1.1. DDE Links

DDE links provide the continuos transfer of real time data from FIXDMACS database blocks to other applications which support dynamic links. Information flow is two way: applications may write or receive data to from the database blocks. DDE links are only available to applications operating within a Micro soft Window's environment on a IBM compatible PC.

4.3.1.2. Historical Data

FIX DMACS provides a facility for the collection and presentation of historical data for future analysis. In FIX DMACS this comprises of three distinct applications: Historical Assign, Historical Collect and Historical Display. Historical data is important for trending and displaying data for long term decision support. This data can ether be utilised within FIX DMACS using the three Historical data manipulation applications or the information can be transported to another application out side FIX DMACS which has been integrated into the FMS control system at a higher level for the purpose of data manipulation and presentation.

4.3.1.3. I/O Devices

This is the channel by which FIX DMACS communicates to the physical manufacturing environment. The actual link involves the definition of an I/O driver which communicates to a programmable device and thus to the actual physical machinery itself connected to the machinery. The I/O diver is a piece of custom written software which converts specific low level device commands and information into FIX DMACS information which can be understood by the database.

4.3.1.4. Operator Screens

Within FIXDMACS there are two applications associated with the operator screens. The first is the draw application the second is the View application. The draw application, creates the operator view screens. The view application views these screens created in draw. This enables a representation of the portion of the factory currently being controlled, to be drawn, then dynamic links can be added to emulate the plants physical movements on the operator screen. The operator screens can be as complicated as one chooses. Careful consideration of the information required on each screen is needed, as the more complex the operator screen the more FIX DMACS scanning and polling resources are required.

4.3.1.5. The Database

The database functions similar to other database software applications in that there are fields in which data is stored for later retrieval by key-words. However the similarities really stop there. The database function in FIXDMACS are built up of functional database blocks. Each database block performs a particular task, and thus

every database block (with the exception of a few) require scan resources and hence processing time. Scan resources are resources that FIX DMACS allocates when collecting or transmitting information as defined by the database blocks.

The creation and interrelationships between database blocks can determine an efficient system from one that is poor performing. It is these database blocks that determine how, when and where data is manipulated along with the data transfer priorities.

#### 4.3.1.6. Network Links

FIXDMACS has its own network architecture to communicate with other remote nodes (a node consisting of a minimum FIX DMACS software installation on a PC) from a common database. This function is very important for the integration of data as it aids in achieving efficient communication of data throughout organisational levels.

FIXDMACS adopts an open system architecture whereby any node on the network may access any database on the network provided it has the required permission. This gives enormous system flexibility.

#### 4.3.1.7. The SAC Task

The SAC task is the Scan Alarm and Control function within FIXDMACS. At current it is set to automatically start up as the system is started. This task determines how often (although not variable by the user) the operator screens are refreshed, what blocks to scan for next (dependent on their scan time) and alarm management within FIXDMACS. The SAC task turns on the dynamics of FIXDMACS; it is this task which manages the supervisory monitoring and controlling functions of FIXDMACS according to parameters set by the database blocks and the supporting applications.

#### 4.3.1.8. SCADA Functions

The SCADA functions of FIX DMACS are central to the operation of the software as FIX DMACS is basically a SCADA package. What a SCADA package does is to replace the original manual control room with a virtual control room based at a PC. This means all the original functions that used to be done manually can now be done by computer software and thus can be more easily controlled. The main SCADA functions of FIX DMACS include:

- Monitoring
- Supervisory control
- Alarming
- Control
- Reporting
- Multi application and distributed communication

4.3.1.9. Distributed Processing capabilities

FIX DMACS has the capability to operate on either a PC as a stand alone node or to operate on a network or LAN as a series of nodes. When operating as a series of nodes on a network, data can be transported around the network, thus allowing the system to become distributed. In this fashion several SCARDA nodes can control portions of a factory at physically separate locations yet information services can be organised to gather data for reporting purposes at one individual node.

The distributed processing capability of FIX DMACS has many great advantages, apart from the advantages outlined in the previous section in reference to a Heterarchical system. One very important advantage that gave the software package tremendous usefulness in relation to the implementation of the FMS control model, was the ability to integrate other packages into FIX DMACS and utilise its network communications capability to allow other applications to gain access to information at remote nodes.

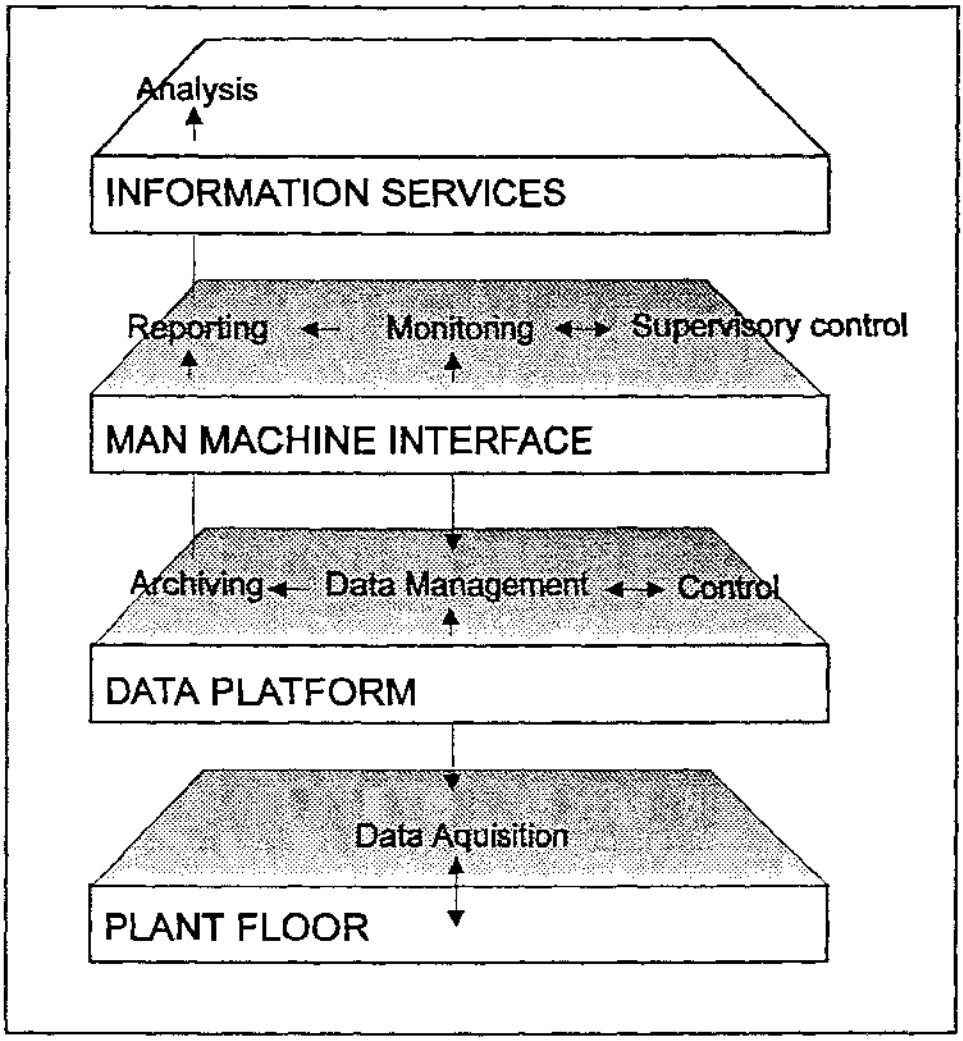



Figure 2
FIX DMACS SCADA Functions

Figure 2 demonstrates the different functions or tasks of the FIX DMACS software package. Monitoring is the ability to display real-time plant-floor data to operators. Supervisory control is the monitoring of real-time data coupled with the ability of the operator to change set points and other key values directly from the computer. Alarming concerns the ability of the software package to recognise in real time exceptional events within the process and to report these events to the operator. As FIX DMACS can operate in a distributed nature alarms can be generated and transported over the LAN to physically separate points. Control is the ability to automatically apply algorithms and adjust process values and thereby maintain a system within set limits. This control goes one step further than supervisory control, in that in some cases it removes the need for human interaction, through using the computer to control the entire process.

The reporting functions are higher information services which can archive or store real time data for analysis, the analysis can ether be done using the reporting functions of FIX DMACS or can be transported to another application via industry standard exchange protocols.

# Appendix 1.2

## FIX DMACS Network Settings

These following settings are not mentioned in the FIX DMACS hardware manual, these settings were obtained through trial and error. They allow FIX DMACS to obtain the necessary resources to operate on the network.

Below is a copy of the config.sys file of the computer which operated the factory control node. It settings were similar to that of other FIX DMACS nodes which operated on the network. There are four parameters which can be altered, these parameters have been highlighted in bold. The first is related to the buffer memory allowed for the Ethernet card, this may need to be increased if problems start occurring. The next three parameters are related to the network resources.

ST = Stations
S = Sessions
C = Commands

It appears that as the number of nodes on the network increases then the number of sessions and commands have to be increased. However if possible these settings should be kept as low as possible so as to allow the network to operate as efficiently as possible. These settings however are only for the network Bios level of the LAN operating in the Department of Production Technology, which is an IBM LAN Manager.

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE RAM i=e000-f7ff /PCC00 X=C800-CBFF X=C800-C9FF
Buffers=10,0
FILES=70
DOS=UMB
lastdrive=Z
FCBS=16,0
DEVICE=c:\len_sys\net-menu.sys c:\len_sys\rem-menu.dat c:\net-menu.flg
DEVICE=c:\len_sys\bootswit.com lz
DEVICE=if 40:F0 1 .eq. 1 { PT Net connection of some sort }
device=     C:\LANMAN\PROTMAN.EXE
DEVICEHIGH /L:1,31888 =C:\LANMAN\SMCMAC.DOS
DEVICEHIGH /L:1,4736 =C:\LANMAN\DXMA0MOD.SYS 001
DEVICE=C:\LANMAN\DXME0MOD.SYS ,16
DEVICEHIGH /L:1,29472 =C:\LANMAN\DXMT0MOD.SYS S=32 C=64 ST=32 O=N
DEVICE=endif
DEVICE=endswitch
COUNTRY=061,,C:\DOS\COUNTRY.SYS
DEVICEHIGH /L:1,15792 =C:\DOS\DISPLAY.SYS CON=(EGA,,1)
rem DEVICE=c:\qemm\loadhi.sys /r:1 c:\dos\ansi.sys
rem DEVICEHIGH=c:\dos\ansi.sys
DOS=HIGH
shell=c:\dos\command.com c:\dos\ /e:2000 /p
rem DEVICE=C:\DRIVERS\MSMOUSE.SYS /G:MOUSE
rem DEVICE=C:\DRIVERS\IBMVGA12.SYS /G:DISPLAY
```

```
rem DEVICE=C:\DRIVERS\IBMPRO.SYS /G:PRINTER
rem DEVICE=C:\DRIVERS\GSSCGI.SYS /T
```

# Appendix 1.3

## Hardware Requirements

The cell controller installation consisted of a single computer running a copy of the FIX DMACS software, described as a full FIX DMACS node. The computer which the cell controller was implemented on was a 486 DX2, IBM compatible PC with a 50MHz clock speed, 250Mbytes of hard disk space and 16Mbytes of RAM. This computer was connected to a Local Area Network via a Ethernet card, the LAN itself was network which operated within the Department of Production Technology. This network consisted of thin coaxial cable, with junction taps at easily accessible points. This network operated using IBM LAN Manager version 3.2 as the network operating system. The network had a single server which contained the network software and a backup of all applications operated in the department. Through the use of this network it was possible for two FIX DMACS nodes to be in contact from anywhere in the department. To allow FIX DMACS to operate on the network in the department a number of alterations had to be made to the network parameters some of which were not mentioned in the FIX DMACS hardware manual.

Operating on the cell controller computer over the top of DOS was Micro soft Windows version 3.11, the Windows application was set up specifically to operate with FIX DMACS, this requires alterations to be made to various Windows parameters as discussed in the chapter on Hardware in section 5.2 of the FIX DMACS manuals. The FIX DMACS application was installed on the computer which acted as the cell controller as per the instructions provided in the FIX DMACS installation booklet. The network capabilities of FIX DMACS were enabled through alteration of the parameters in the configuration application provided with FIX DMACS, this was done as explained by the FIX DMACS installation booklet. Once FIX DMACS was installed in the cell controller computer it operated within the WINDOWS environment.

# Appendix 1.4

## FIX DMACS Database Block Information

FIX DMACS provides 30 different types of blocks which can undertake a variety different tasks. Database blocks within the VMD consisted of a number of different types, depending on the function to be performed. For example a block which would be used to collect information about the speed of a lathe spindle, would most probably be an Analog input block.

V1MACHBUSY, belongs to VMD 1 group type and has the unique name of MACHBUSY which is also an abbreviation of its function MACHINE_BUSY. The Block Type is a two letter abbreviation of the type of database block, eg AI Analog Input and AO Analog Output blocks. The Block Description is a full description of the group, type and function of the database block. The first part denotes its block group type, in the case of the examples this is VMD 1. The second explains the type of block, SO, this stands for status object. A status object is a data object which reflects the status of the low level device. The most common types of blocks are status objects SO and command objects CO. Command objects are objects which transport the intentions of the control algorithm to the low level device. The third and final part of the block description is the data object name. This is the particular task or piece of information which the database block is related to.

# Appendix 1.5

## The Virtual Manufacturing Device

A VMD provides the cell controller with the ability to effect the operation of and to gain current state information of a low level device. The basic concept behind the VMD is to provide a virtual representation in software, of a physical machine. In the case of the FMS control system it was important that the VMD was generic, ie it was able to be used to control any number of different low level devices without alteration of the structure of the VMD. A portion of the core VMD services were implemented within FIX DMACS.

The building blocks of FIX DMACS were used to combine a number of individual elements which had the capability to reflect the state of a low level device. FIX DMACS Database blocks were used for this purpose, each block representing a particular piece of VMD information. These blocks could then be linked via a low level control device, such as a PLC to a real low level device such as a CNC lathe.

From the specification of MMS, a number of services were highlighted that would be applicable as generic services to be provided by a VMD to the cell controller. These services would provide the control and information gathering ability for the cell controller. A core contingent of services would provide the majority of operations required to effectively control any low level device. However a number of custom services would need to be provided for a small number of operations. It was felt that these services should also be supported by the VMD.

These services were then implemented within the VMD. This was done through allocating a single database block of a suitable type to a service or a portion of information about a service. These individual blocks were called data objects, as they are objects which represent the state of a low level device. A number of related objects were called a data object group. The most common type of database blocks that was used to implement a particular data object were Analog input and Analog output blocks (refer to FIX DMACS manual).

The cell controller can control up to four different low level devices, thus each cell controller will contain four VMDs one for each of the low level devices. Each of these VMDs does not have to be used and can sit unobtrusively in an idle state. The contents of the VMD can be altered through three different channels. The first is by a operator con figuring each of the devices in all cells within the factory. The effect of the configuration will be to alter the contents of the VMDs within the cell to reflect the physical devices which the cell controls.

The second way in which the information within a VMD could be altered, is through the VMD updating its current state to reflect changes in the state of the low level device. This is achieved by linking FIX DMACS database blocks to area in the memory of the low level device controller. The device controller which is directly connected to the low level device will then determine via its sensors what the current state of the low level device is, then write that information to the relative addresses in memory. FIX DMACS will then use its I/O driver software to read the information from the memory within the low level device controller then place the appropriate information in the relative database blocks. The third way in which information in the VMD can be altered is by the intervention of the device control algorithm which is based in FIX DMACS.

# Appendix 1.6

## Limitations of FIX DMACS Programming Blocks

The programming language itself contains 31 different commands capable of performing a variety of different functions from data manipulation to the opening of Windows applications. The programming language itself was relatively comprehensive, it did however have several major limiting factors.

1. A limit of 19 instructions per program block.

2. Transferring data from one block to another required two instructions.

3. Program blocks were unable to transfer information between database. Thus inter-database communication was difficult at this level.

4. The use of program blocks slowed the operation of FIX DMACS considerably.

5. Very limited debugging features meant that locating errors within a program was a difficult process.

The first of these problems was probably the most limiting factor of all. A maximum of 19 lines of instructions of code combined with an inefficient syntax meant that this limit could easily be used, writing even the most simple program. To overcome this short fall it was possible to call one program block from another, just as one would call a function from within a C program. This enabled a complicated series of task to be performed through using a number of program blocks. To facilitate in the construction of an easy to understand device control algorithm, a single master control algorithm was written that would call a secondary program block which would in turn undertake the required operations associate with that step. This is a form of instruction decomposition which allows a series of relatively complicated operations to be performed by simple programmable units.

# Appendix 1.7

## Bid Formulation

The bid formulation or bid calculation can occur at the cell controller in response to a BID_REQUEST. A bid request is generated by either the factory controller releasing a new job into the system or by a cell re-bidding a partially completed job into the system, in either case a BID_REQUEST is received by the cell controller and the bid formulation process begins. The BID_REQUEST and subsequent bidding information is transported by the Bidding information transportation entity. This entity will be discussed latter in full, however it is the entity in the cell controller which retrieves the BID_REQUEST information and the appropriate bidding information from the Factory controller after the BID_REQUEST has been received. Bidding information is transported upon the LAN, this allows FIX DMACS to transfer information from point to point via the network.

Once the Auction control entity receives a BID_REQUEST and soon after the appropriate bidding objects are obtained it can then set about formulating a response. The transportation of the bidding objects from the factory controller to the cell controllers is always done via a view screen written within FIX DMACS. The first step once the bidding objects have been received is to decide if the cell is capable of undertaking the task at the centre of the auction ie the auction task. It determines this by comparing information about the auction task, ie Bidding data object group with the capabilities description of the low level devices, ie the VMDs. This process first checks to see if the cell has a low level device capable of performing the task, it then checks to seen if that device has the necessary resources to undertake the auction task. This is achieved by comparing the description of the auction task contained within the bidding objects with the resources available in the capable VMD. A calculation block is then used to calculate a bid. The calculation block is a FIX DMACS database block which can be defined to retrieve data from other database blocks and combine them according to a mathematical formula. The Bidding calculation is as follows:

$$\begin{pmatrix} Current\_QueueTime + (No.\_of\_Cycles \times (CycleTime + \\ (LoadTime + UnloadTime)) + SetupTime) \end{pmatrix}$$

This calculation takes the Cycle time of the auction task adds to that the load and unload time of the auction task, then multiplies that by the number of cycles in the auction task, then adds to that the set up time of the low level device and the length of time currently in the incoming queue. This provides a bid which is an assessment of the earliest possible time to which the cell could complete the part. This assessment of the Earliest Finishing Time (EFT) is then transported back to the factory controller accompanied by an acknowledgment that the bid has been made. Once the factory controller has collected the bids it can then make a decision on which cell will be awarded the job.

When the winning cell has been informed that it has been awarded the job at the centre of the auction, it then retrieves the current queue data object group, from its VMD, this information was included with the Bidding Data object group. This information is then PUSHED on to the current queue, the PUSH operation places the current queue information onto the end of the current queue of the device which won the auction in the cell.

Co-ordination of the bidding process was required as only one auction could take place at one time. Therefor a polling system was created where each cell and the factory controller in the FMS system took terns in undertaking an auction. Thus a cell requiring to auction a job it has just completed would continue working on its next job until it was polled, then it would undertake an auction and send the job information and work piece to the winning cell.

*Re-bid Control*

The re-bid control function is a FIX DMACS program block, this program block has the task of bidding either the partially completed current task or bidding the next task in a partially completed job. These two cases occur either when a task is only partially completed due to a device error, or when the current task in a job is completed and the next job is required to be processed. The process of initiating the auction or re-bidding is the same in both cases, the only difference is the source of the bidding objects. Where an error has occurred part the way through the completion of a task and the task is unable to be completed, then the source of the bidding objects would come from the current task, ie from the VMD. This can occur during the operation of the device control algorithm. If an error occurs whilst the low level device is processing then the bidding objects are transported from the VMD and pushed on to the next queue to be re-bid. This transportation is done via DDE links established by the Next_Queue application.

In the case where the current task has been completed, the next task from its job file must then be re-bid. If their is a next task which requires re-bidding then the bidding objects are retrieved from the appropriate job file (IFFile). This occurs during the set down cycle of the device control algorithm, where the current task of the current job is completed. The device control algorithm then instructs the Next_Queue application to retrieve the current job file, then to increment the current task counter by one. The bidding objects are then retrieved by the Next_Queue application from the task which is being pointed to by the current task counter, and then are pushed on to the next queue.

The re-bid control algorithm holds in a wait state until it is polled by the bid coordinating entity. When the re-bid control algorithm is polled it receives a bidding token over the network, this bidding token can only be at one cell at one time and thus serves to limit the bidding process so only one auction can be held at any one time. Thus whilst the cell has the bidding token it can request an auction which will not conflict with any other auctions at that time. As soon as the re-bid control algorithm receives the bid token it then checks each of the four next queues (one next queue for each low level device) to see if they have any tasks that require auctioning. If a a device has a task which requires to be auctioned the re-bid control algorithm

then takes the bid token so no other cell many request an auction whilst it is conducting its auction. Now the auction process begins.

The next step in the re-bid algorithm is to request the Bid_Rotation application (the application which provides the bid polling and some of the bid transportation functions) to transport a bid request along with the bidding objects, popped from the top of the next queue to the factory controller bid decision entity. The re-bid algorithm does this by transporting the bid request and bid objects from the re-bid control algorithm where the bid token is currently being held, to the Bid_Rotation application. This is done by DDE links established between the Bid_Rotation application and the cell controller. This application then transfers these objects to the auction decision entity, the factory controller, via DDE links. The auction control entity responds in the same fashion when receiving a bid request and bidding objects irrespective of the source. As bid requests can come from any cell in the factory and from the job release queue (this is the queue which releases new jobs into the system) at the factory controller. Once the auction decision entity receives the bid objects and the bid request it then broadcasts these upon the network and waits for a response. Once it receives all the bids it then decides which cell will win the auction.

*Auction Co-ordination*

The auction co-ordination entity is responsible for the polling of the factory controller and other cell controllers. This polling serves as a method of limiting the number of entities who are currently undertaking an auction. Due to the way in which the auction process has been implemented using FIX DMACS it is possible to only hold one auction at once. Thus the polling system was devised which polls each of the necessary entities within the factory in turn. This allows only one entity at any one time to be able to use the auction system. It was impossible to implement this sort of polling method within FIX DMACS as the program blocks which would have to be used in this case are unable to access information in blocks in other databases. This was achieved through the creation of an application which was written in Visual Basic. An application written in VB was able to gain access to information in all cell controllers databases, as well as the factory controller. This was achieved through the use of DDE links. By using DDE links to provide an information path between the application and the destination database it was possible to achieve point to point communication with all databases on the network.

An application was written called Bid_Rotate, this application had the task of moving a bid token around all the capable entities within the network. Bid_Rotate operated through using three DDE links to a database. The first DDE link was called the bid operate link. This link was established when the application was first opened. When this occurred Bid_Rotate firstly opened a file on the network drive called nodefile.txt, this is a file defined by the system operator which contains a list of all the addresses of all the nodes in the factory controller. This list includes the addresses of the FIX DMACS databases of all cell controllers and the factory controller. This enabled the Bid_Rotate application to gain access to information in all databases in all cell controllers and the factory controller, on the network. Once the Bid_Rotate application had open this file it then knew the addresses of all nodes of the network. It then, one node at a time, established a link between itself and a database block on

each node, called "bidoperate", it then set the value of this block to 1. Once this had been completed all cells and the factory controller had a database block called bidoperate set to 1.

This was done because every node on the network contained a copy of the Bid_Rotate application and each was instructed to open and start operating if the value of the bidoperate block in the local database was set to 0. This meant that the first node to be opened and to start operating would also have the Bid_Rotate application operating at that node. All other nodes would simple open as normal. Each cell also contained a program block within its database that would check every so often to see if the Bid_Rotation application was operating if it was no longer operating properly it would reset the value of its Bidoperate database block. This means that if for some reason the Bid_Operate application failed or a cell failed whilst it held the bid token and thus did not release it back into the system, then the system would reset itself and a new copy of the Bid_Rotate application would open, thus starting the process again. This is simply a way of making the Bid_Rotate operation recoverable in the face of errors.

The second two DDE links were called bid token and bid enable, these two links were established then unestablished with every node on the network on a regular basis. The Bid_Rotate application establishes a link the first node listed in the nodefile.txt then waits for a period of time then moves on to the next node listed in the nodefile. However the Bid_Rotation application will only move on to the next link if the current node does not required the bid token, else it waits until the node has finished.

The first of these two links, bid token is established with a database block in all nodes on the network called "bidtoken" this database blocks has a value of 0 normally and 1 when it contains the bid token. Thus when the Bid_Rotation application establishes a link with a node it sets the bidtoken database block value to 1. It then waits for a period of time, (this can be varied) in the case of the FMS controller implementation this wait time was set to 5 seconds. If the local cell has jobs it wishes to hold an auction for the local re-bid entity will set the "bidenable" block to 1 to prevent the bidding token from being moved on to the next cell in the factory. Once the time period is up the Bid_Rotation application then checks the value of another link, called bid enable. The bid enable link is established with a database block called "bidenable". If at the end of the wait time the value of the "bidenable" database block is 0, then the bidtoken block is set back to 0, then links between Bid_Rotation and the cell are de-established and then re-established with the next node on the nodefile list. This has the effect of moving the bid token around all nodes within the factory because only one node anywhere in the factory will have its bidtoken database block set to 1, the rest will all be set to 0.

When a node has its bidtoken database block set to 1, its re-bid control entity will then check its next queues to see if any jobs require to be rebid, if so it then sets the bidenable database block to 1 before the wait time is up. That node then has exclusive use of the auction process and will then hold an auction of up to one job from each of its next queues, (maximum of four next queues). Once the re-bid entity has checked all of its next queues and held an auction for each of the jobs at the top of

the queues it then sets the bidenable database block to 0, this will allow the Bid_Rotation application to move the bidding token on to the next node.

# Appendix 1.8

## The Auction Co-ordination Entity

Two links are formed between the Visual Basic auction co-ordination application and the database.

The first of these two links, bid token is established with a database block in all nodes on the network called "bidtoken" this database blocks has a value of 0 normally and 1 when it contains the bid token. Thus when the Bid_Rotation application establishes a link with a node it sets the bidtoken database block value to 1. It then waits for a period of time, (this can be varied) in the case of the FMS controller implementation this wait time was set to 5 seconds. If the local cell has jobs it wishes to hold an auction for the local re-bid entity will set the "bidenable" block to 1 to prevent the bidding token from being moved on to the next cell in the factory. Once the time period is up the Bid_Rotation application then checks the value of another link, called bid enable. The bid enable link is established with a database block called "bidenable". If at the end of the wait time the value of the "bidenable" database block is 0, then the bidtoken block is set back to 0, then links between Bid_Rotation and the cell are de-established and then re-established with the next node on the nodefile list. This has the effect of moving the bid token around all nodes within the factory because only one node anywhere in the factory will have its bidtoken database block set to 1, the rest will all be set to 0.

When a node has its bidtoken database block set to 1, its re-bid control entity will then check its next queues to see if any jobs require to be rebid, if so it then sets the bidenable database block to 1 before the wait time is up. That node then has exclusive use of the auction process and will then hold an auction of up to one job from each of its next queues, (maximum of four next queues). Once the re-bid entity has checked all of its next queues and held an auction for each of the jobs at the top of the queues it then sets the bidenable database block to 0, this will allow the Bid_Rotation application to move the bidding token on to the next node.

# Appendix 1.9

## Bidding Information Transportation Entity

This implementation utilises the network capabilities of FIX DMACS to transport the Bidding objects from one node to another. This was achieved through using the macro programming language within FIXDMACS view screens. This macro language allows the designer to automate objects within a view screen, through defining a series of actions that can occur in response to a change in the database. For example a macro may be attached to a view screen which shows liquid flowing through a pipe, this macro could sense via the database, when the flow rate through the pipe becomes critical and then inform the operator of this fact. Macro's written for a particular view screen can be activated either by user interaction with the objects in the view screen via the mouse or keyboard, or through the opening of the view screen to which it was attached. One very useful function available via this macro language is the ability of the macro's to access information at any database upon the network, something that program blocks are unable to do.

During the normal course of the operation of the cell controller and number of view screens will be activated, this will allow the operator at that cell to gain important information about the state of all devices within that cell. These view screens which can be selected by the operator will show a variety of information, such as the current state of devices, current level of in and outgoing queues. A macro program was written and attached to all view screens that could be open at any one time at the cell controller. This meant that no matter what information the operator had currently selected to view, the macro would still be operating. The logic of the macro meant that soon after it began to run it went into a loop and would only exit from the loop at the occurrence of one condition. This macro was designed to wait until a Bid Request was broadcast upon the network and then retrieve the Bidding objects from the factory controller. Thus the macro would only exit the initial loop in the condition that a Bid Request had been broadcast upon the network from the factory controller and at that point it would begin to retrieve all the bidding objects from the factory controller. This cycle was continued until the Bid Request was withdrawn from the network at that point it would return into its initial loop and wait for the next Bid Request to be broadcast.

# Appendix 1.10

## Incoming and Outgoing Queues

A queue consists of a list of items with each item containing a number of fields, the fields contain the individual pieces of information that are required for either the next or current queues. Each item in the current queue only contains a few fields, these correspond to the Current Queue data object group. The incoming queue is only required to store the name of the IF File which contains the information about the task at hand. However each item in the next queue contains a number of fields, these correspond to the next queue data object group. These fields must contain all the information required to re-bid the task in the current job, directly after the one just completed. The queue itself consist of a number of items placed in a sequential order, a number of operations can be performed upon the items within the queue. The items within the queue can either be retrieved from the top queue (POPed) or items can be placed at the start or end of the queue (PUSHed). The queue operates on the first in first out basis, only the items at the top of the queue can be access or POPed and once the top item has been POPed then the next item in the queue comes to the top. These queues implemented within the Next_Queue application can contain up to 100 items, if this limit is exceeded then the items at the top of the queue are dropped off in order to make room for those at the end.

The transfer of information from the current queue and to the next queue is undertaken by the device control algorithm within FIX DMACS. Pushing new jobs onto the current queue is undertaken by the auction control entity at the cell, this is then entity which places bids for jobs being auctioned, if it wins a job it then pushes that job onto the current queue of the appropriate device within the cell. The popping of jobs from the next queue is undertaken by the Re-bid control entity, this is the entity which is responsible for Re-bidding the jobs once the current task within the job has been completed. The device control algorithm determines when a new task is to be started and when the current task has completed. The operation of the queues is controlled by a DDE link between a FIX DMACS database block and the Next_Queue application. The device control algorithm alters the value of a database block called "IFCONTROL" to perform one of two operations.

1. To push information from pre defined Data base blocks on to either the next or current queues.
2. To pop the job name from the current queue then to load the IF file pertaining to that job and transfer the job information into the respective VMD.

The next queue can also be controlled by the Re-bid algorithm which pops completed jobs off the top of the next queue then re-bids them into the system. The DDE links required for all this information transfer are established between the database blocks within the respective VMD and the copy of the Next_Queue application pertaining to that VMD. Each Next_Queue application contains only one next and current queue. In each cell there can be up to four VMDs and therefor up to four Next_Queue

applications. Two different types of DDE links are used in the implementation of the Next_Queue application, manual and automatic links. Automatic links are used for the control links and manual links are used for the information transfer links, automatic links provide a information link which is automatically updated up Windows however manual links require information to be "requested" or "posted". This allows the Next_Queue application to sit in the back ground, monitoring the automatic control link until it is modified by one of the entities within the cell controller and is required to perform a pop or push operation. Like all the Visual Basic applications that make up the cell controller, Next_Queue operates as a back ground task.

# Appendix 1.11

## The Device Control Algorithm

Once the CAD/CAM operation has been undertaken by the operator a series of generic machine part files exist. However these files are not specific to any particular low level device and require post processing for a specific application. To perform this post processing the operator must know which specific device in the factory the machine file will be required for. Unfortunately due to the opportunistic scheduling method one can never be certain where a specific task will end up to be processed. Therefor it was necessary to produce generic machine data files and then give each cell the ability to post process the generic code into a form which the specific low level device can interpret.

The task of the data transportation entity is to take the machine data file after it has been post processed by the cell and to transfer it via a communications port available on the computer to the low level device concerned. For this task an application was written in Visual Basic, this application interfaced within FIX DMACS via DDE links. These links enabled the DCA to control the operation of this application. Three DDE links were used, one to provide operational control, another to inform the application of the file name to be transferred and another to specify the communication parameters.

The device control algorithm in the set up phase would first post process the machine data file and then instruct the transfer of the file name of the post processed file and the communications parameters to the Visual Basic application (FIX_COM). Next the device control algorithm would alter the value of the control link to instruct the FIX_COM application to down load the machine data file. This down load was conducted in the manner specified by the communications parameters. The FIX_COM application loaded the file from the network drive, then down loaded it at the specified communications rate and out via the correct RS-232 communications port on the cell controller. The communications port specified was the one which was connected to the low level device controller. Once the machine data file had been successfully transferred the actual processing of the work piece could begin.

# Appendix 1.12

## Intelligent Entity

For example the instruction issued from the cell controller may be to START, by this it means to start the low level device operating. The generic decision rules decide that to "run" the low level device a number of operations are required, these being WARM_UP, CLOSE_INTERLOCKS and SWITCH_ON. These operations may differ depending upon the current state of the low level device, if the low level device had just completed processing a previous part then it may not require the WARM_UP operation, how ever in this case it does. These commands are then interpreted by the specific implementation routines and the PSC goes about using its actuators, relays, sensors and switches to undertake the above commands.

## Appendix 1.13

## Suggested Method of Implementing the Two Layered IE Model within the PSC

The best way of implementing this model would be through the intelligent use of the loops available to the PSC.

One loop could be used as the start loop to start the I/O routines and any other loops that require constant monitoring, such as I/O driver program loops. This loop can then hand the focus of the program over to a main loop.

The main loop monitors the command objects within the VMD and if an instruction is received then this loop calls a generic decision loop specific to that command.

This generic decision loop then makes a decision based upon the logic of the loop and any information required. This loop then calls a number of other loops, each which undertake one of a number of specific implementation tasks, hence instruction decomposition.

The Specific instruction loop then simple implements the task it has been designed to undertake.

Further loops will be required to monitor the I/O states and report changes back to the VMD.

# Appendix 1.14

## Node/Factory Definition

The node name of a node is an individual name given to each FIX DMACS database. As there can be only one database per node and each cell within the factory has one FIX DMACS database operating upon it, this node name was the individual name chosen for each cell controller. The node names enables FIX DMACS to distinguish between the different databases upon the network, it is like an address. Because the structure of all cell controller databases are essentially the same, knowing the node name of a particular cell upon the network will enable information within that cell controller to be accessed. The operator uses the node definition window to enter in all the nodes within the factory into a list. This list allows the factory controller to gain access via the node name to the databases contained at each of the cell controllers. This list is stored in a file called "nodelist.txt" upon the network drive, thus enabling any computer/node on the network to gain access to this file.

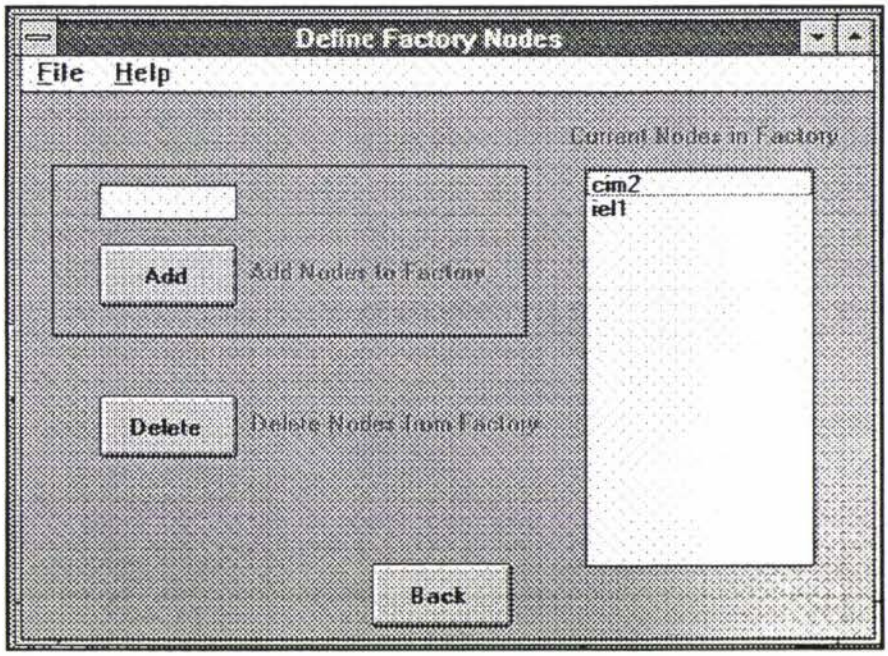



Figure 1
Node definition window

Once the operator has entered in all the node names of all the cell controllers within the factory this window can then be exited an the user can then move back to the menu shown if Figure 1 (Factory definition menu window), the factory definition menu window. The operator can then select the define factory layout option from the menu.

# Appendix 1.15

## Factory Definition

Once the user has select a node from the list to define, a further window is opened this window is called the cell definition window and allows the operator to define each up to four of the low level devices in each cell. The user is presented with a window which allows the user to enter all the information required to define a particular device. Once this has been completed the user can then move onto the next record, and do the same with up to four records each corresponding to one of the devices in the cell selected. Presently the user is able to define for each device in a particular cell, the device name and type, the set up time, down time, service interval and which communications port it is connected too. Once the user has finished defining a particular cell the cell definition information is written to a file with the same name as the node plus a ".txt" extension. The user can then continue on with this process to define the configuration of all cells within the factory.

When this has been completed the user can then move back to the factory definition window. From here the last option the user can select is to configure the factory. This option causes the cell definition information in the node files to be transported to the cells them selves, in effect this operation configures the factory. When the factory configuration button is selected the application opens the nodelist.txt file and uses the list to find the file names of the cell configuration files. The application starts at the top of the list and opens the cell configuration file which has the same name as the first node name in the node list. Once this file has been opened the cell configuration information that was defined by the user, is down loaded one device at a time to each of the up to four VMDs within the cell controller. This process is repeated for each node on the list. This effectively causes the VMD to take on the properties of the device defined by the user. The respective VMDs within the cell will now contain the device names, types, set up times etc, defined by the operator. This has the effect of defining the Virtual Factory, however this does rely on the operator matching up the definitions of the devices within each cell to, the actual physical devices connected to that cell.

# Appendix 1.16

## CMSA/CD Protocol

The Carrier Sense, Multiple Access with Collision Detect (CSMA/CD) is the protocol used in the IEEE 802.3 international standard, this standard is Ethernet. The standard baseband system operates on 10Mbps using coaxial cabling or on twisted pair. (Tanenbaum 1989). Ethernet LAN is the ideal choice, it is both easy to use and cheap as well as becoming increasingly common. CSMA/CD is an excellent protocol for general network traffic with light to medium traffic density, however in a heavy traffic density environment time critical information can be delayed possibly rendering it useless.

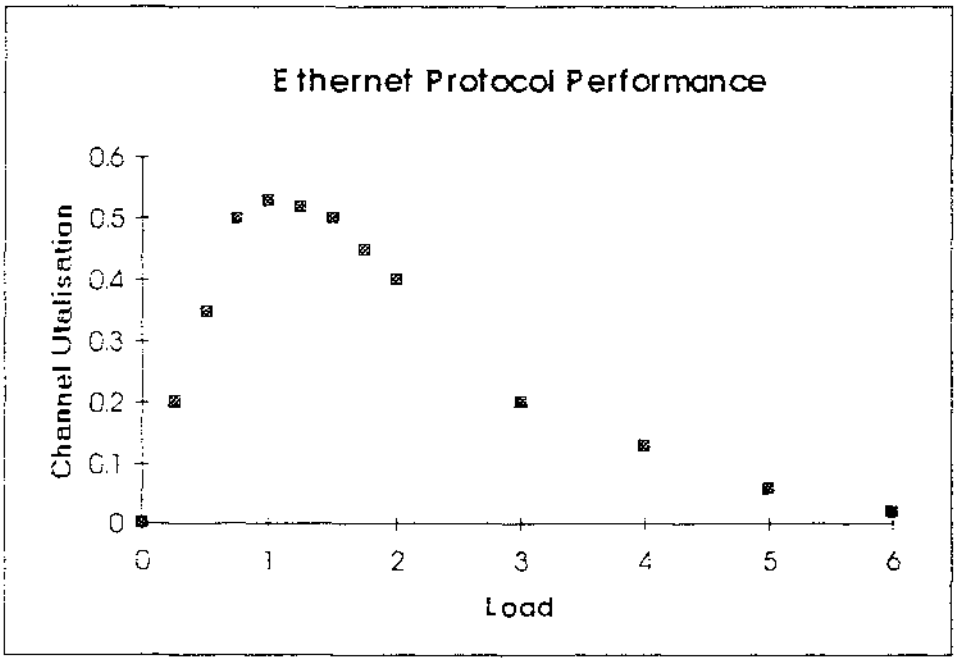



Figure 1
Channel Utilisation vs Load for the Ethernet protocol
(Tanenbaum 1989)

The CSMA protocol tends to degrade in efficiency as the network load increases in an exponential fashion as shown in figure 1.

Ethernet LANs are very common, thus it was decided that basing the FMS control system on an existing and successful communications network would it applicable to a wide range of environments. The Ethernet LAN in the case of the FMS control system was used to connect the factory controller with the cell controllers and do so in a BUS network The BUS network with the Ethernet protocol allows all nodes on the network equal access to the communications resource. It allows a tree like installation (Shaw 1987) of nodes permitting relatively easy installation and expansion of the network, also nodes can be added or subtracted from the network without

disturbing the rest of the network. The BUS topology which the Ethernet LAN uses is also generally more reliable than other network topology's such as star or ring networks (Tanenbaum 1989).

## Appendix 1.17

# FIX DMACS Communication

The following sections detail the structure where possible of the each of the eight different forms of communication.

These descriptions of information transportation is important as they give descriptions of FIX DMACS database blocks and Visual Basic data structures and the links between them.

### 5.5.2.1. File Management

File management is a task of the Visual Basic application which must manage data files from both CAD and CAM packages. The communications link between the front end Visual Basic application and either of the CAD/CAM applications has yet to be formally defined. However it is reasonable to say that the communications link will be provided in the form of data files being produced by either of the applications and stored in the appropriate location on the network server. The file management task will require the data files to be in certain formates in order to be transferred between the CAD/CAM applications.

### 5.5.2.2. Machine Data Information

The machine data information is information which is transferred between the Visual Basic application and the low level device. Once a part has been designed by the CAD application then the appropriate manufacturing plan has been specified by the CAM application and a machine data file for a particular low level device has been created, this file must then be transported to the appropriate low level device. The machine data file contains the program which the low level device will use to manufacture the part designed by the CAD application. The device control algorithm instructs a Visual Basic application (File_Com) to transport the appropriate machine data file from the network server to the appropriate low level device at the specified communications parameters.

This application then transfers the machine data file via one of the RS-232 communications ports on the computer which the target low level device is connected to. The actual specifics of the data transfer vary for different low level devices as different low level devices have different communications requirements, thus the Baud rate, Comm port number, Hand shaking protocol, Error checking protocol and Stop bits are all varied. The physical wiring which connects all the low level devices to the local cell controller is the only part of the data transportation process which is fixed. The physical wiring of this connection is done in accordance to the RS-232 protocol, reference to this protocol can be found in the FIX DMACS hardware manual (ref FIX DMACS hardware manual).

5.5.2.3. PSC - FIX DMACS Communication (Control Data)

The PSC to FIX DMACS communication, is concerned with the transportation of control data or control information from the PSC (low level device controller) to FIX DMACS and visa versa. This transportation of information as discussed earlier is handled by the I/O driver, the I/O driver is software which connects FIX DMACS to the low level device controller, or in this case the PSC. In general terms the I/O driver allows information of any type to be passed to and from the low level device controller. However in the case of the FMS control system only information of a certain predefined type is valid for this communication. Valid information is passed across the communication link, from specific FIX DMACS database blocks to specified addresses in the low level device controllers memory. Each of the individual pieces of information which flow across the link have a definite source and destination, and can be only one of two types, either control information or machine data. Control information flows from FIX DMACS to the low level device controller whilst machine data flows from the low level device controller to FIX DMACS. Control information consists of instructions issued by the device control algorithm within the cell controller to the low level device controller, where as machine data is information about the current state of the low level device itself.

Weather or not the information passing across the link is either control information or machine data, it represents a service that is provided by the low level device controller to the device control algorithm. This service could be the collection of status data or the execution of a command. The set or group of services provided by the low level device is called the Virtual Manufacturing Device. The VMD consists of a number of individual services, represented at the FIX DMACS level as a number of database blocks. As yet the VMD has not been completely defined, however a partial VMD has been defined so as to allow the provision of basic services to the device control algorithm. Thus the form of the communications interface between the low level device controller and FIX DMACS (cell controller) is defined by the VMD.

Table 1
VMD Services Description

| Database Block Name | Block Type | VMD Service Description |
|---|---|---|
| V1LOADCOMP | AI | VMD1 SO LOADDEVICE_COMPLEATED |
| V1MACHBUSY | AI | VMD 1 SO MACHINE_BUSY |
| V1MACHERR | AI | VMD 1 SO MACHINE_ERROR1 |
| V1MACHIDLE | AI | VMD 1 SO MACHINE_IDLE |
| V1MACHRUN | AI | VMD 1 SO MACHINE_RUNNING |
| V1MACHSTOP | AI | VMD 1 SO MACHINE_STOPPED |
| V1PROGREDY | AI | VMD 1 SO PROGRAM_READY |
| V1UNLOADCM | AI | VMD1 SO UNLOAD_COMPLEATE |
| V1COMMID | AO | VMD1 SO COMMUNICATION_IDPARAMETER |
| V1COMMPORT | AO | VMD1 SO COMMUNICATION_PORT |
| V1CURCYCNO | AO | VMD1 SO CURRENT_CYCNo |

| V1CYCCOMPL | AO | VMD 1 SO CYCLE_COMPLEATETIME |
|---|---|---|
| V1CYCDUR | AO | VMD 1 SO CYCLE_DURATIONTIME |
| V1CYCNO | AO | VMD1 SO No_OFCYCLES |
| V1CYCSTART | AO | VMD 1 SO CYCLE_STARTTIME |
| V1EMERSTOP | AO | VMD1 CO EMERGENCY_STOP |
| V1FILETYP | AO | VMD1 SO FILE_TYPEREQUIRED |
| V1FINALTSK | AO | VMD1 SO FINAL_TASK |
| V1LOAD | AO | VMD 1 CO LOAD_DEVICE |
| V1LOADUNTM | AO | VMD1 SO LOAD_UNLOADTIME |
| V1MACHDOWN | AO | VMD 1 SO MACHINE_DOWNTIME |
| V1MACHINIT | AO | VMD1 CO INITIALISE_DEVICE |
| V1MACHSERV | AO | VMD1 SO MACHINE_SERVICEINTERVAL |
| V1MACHSET | AO | VMD 1 SO MACHINE_SETUPTIME |
| V1MACHTYP | AO | VMD 1 SO MACHINE_TYPE |
| V1NXTQUENO | AO | VMD1 SO NEXT_QUEUENUMBER |
| V1PARTSER | AO | VMD1 SO PART_SERVICINGMETHOD |
| V1PARTSET | AO | VMD1 SO PART_SETUPTIME |
| V1PROGDOWN | AO | VMD 1 CO PROGRAM_DOWNLOAD |
| V1PROGUP | AO | VMD 1 CO PROGRAM_UPLOAD |
| V1QCONTROL | AO | VMD1 CO QUEUE CONTROL |
| V1QUENO | AO | VMD1 SO QUEUE_NUMBERIN |
| V1QUETIME | AO | VMD 1 SO QUEUE_TIME |
| V1QUETOP | AO | VMD1 SO QUEUE_TOPOF |
| V1START | AO | VMD 1 CO START_DEVICE |
| V1STOP | AO | VMD 1 CO STOP_DEVICE |
| V1TOOLAVL | AO | VMD 1 SO TOOLS_AVAILABLE |
| V1UNLOAD | AO | VMD 1 CO UNLOAD_DEVICE |
| V1COMMPAR | TX | VMD 1 SO COMMUNICATION_PARAMETERS |
| V1FEEDSIG | TX | VMD 1 SO FEEDBACK_SIGNAL |
| V1FIXREQ | TX | VMD 1 SO FIXTURES_REQUIRED |
| V1GEOFILE | TX | VMD 1 SO GEOMETRIC_FILE |
| V1IFFILE | TX | VMD 1 SO IFFILE_NAME |
| V1JOBNAME | TX | VMD 1 SO JOB_NAME |
| V1MACHFILE | TX | VMD 1 SO MACHINEFILE_NAME |
| V1MACHNAME | TX | VMD 1 SO MACHINE_NAME |
| V1TOOLSREQ | TX | VMD 1 SO TOOLS_REQUIRED |

#### 5.5.2.4. Inter-software Communication and Network Communication

Inter-software communication is the communication that occurs between two different applications in the case of the data model this refers to communication that occurs between the Visual Basic front end program and FIX DMACS as well as communication between one of the several Visual Basic applications that operate at the cell controller, and FIX DMACS. Network communication is the transportation of information over the network, via FIX DMACS. Often communication between two different applications, ie inter-software communication occurs over the network,

thus inter software network communication can occur. However in both of these cases of communication the type of information and the form must be predefined. Each of the entities participating within the communication must have knowledge of the communication interfaces of the other entities it wishes to communicate with. For this reason the form in which information is transported has been defined, these definitions are called data object groups. Data object groups define the interfaces between various entities within the system, further more data object mapping's are also required in order to fully explain the transportation of information from one entity within the system to another, regardless weather network communication is involved or not.

For every transportation of information between entities with the FMS system at present there is a corresponding data object group and a data object mapping which demonstrates the path through which the information flows.

The information transportation as discussed previously can be undertaken in either one of three ways. 1. DDE transfers, this transports information via Dynamic Data Exchange between FIX DMACS and a Visual Basic application. 2. Transfers within a FIX DMACS node or cell, this is achieved by coping data from one block to another. 3. Transfer between FIX DMACS nodes via the view screen, this uses FIXDMACS own programming lanugo to transfer information between two data base block.

### A. JobRecord Data Object Group or Job Information data object group

This data group transports Job information between the factory controller and the cell controllers. The factory controller places the job information into a data structure called JobRecord, this is then written to file. The file is then opened at a latter date by the cell controller and the information is extracted and transferred into the cell controller database.

| Factory Control (VB) | Data Structure (File) | Cell Control (FIXDMACS) |
|---|---|---|
| Name of Job | Jobname | Jobname |
| Estimated cycle time | Estcycletime | estcyctm |
| Set up time | Setuptime | Partset |
| Machine data file name | Machinefilename | Machfile |
| Geometric file name | Geometricfilename | geofile |
| Fixtures information | Fixturesfilename | fixreq |
| Tooling Information | Toolsrequired | toolsreq |
| Final Task | Finaltask | finaltask |
| Number of cycles | Noofcycles | cycno |
| Estimate load/unload time | Estloadunloadtime | loaduntm |

These data objects are used for the control of the information transportation and thus have no mapping onto the cell controller, they simply represent a information management overhead:

| | |
|---|---|
| (Task index number) | Currentindex |

| | |
|---|---|
| (Total number of tasks) | Tasknumber |
| Task Name | Taskname |
| (Release control) | Duedate |
| (Queue force control) | Force |
| (Job operation control) | Status |

## B. Bidding Information Data Object Groups

The Bidding information data object groups are a series of groups which are used to transport bidding information around the FMS control system, because bidding information is required to flow through a variety of different paths a number of mapping's are required to demonstrate how the bidding data object group is transported to different entities.

### 1. Bidding data object group maps

This group maps bidding information between the cell controllers bidding entity and the cell controllers next or out going queue data object. The bidding information within the cell controller FIX DMACS database are transferred from the bidding data object group, which consists of a number of database blocks, to the next queue data object group. This transfer of information is in order to place the next task bidding information upon the next or out going queue. This then frees up the bidding data object group so new information can be transferred to it without distorting the old information.

| Bidding data object group | Next queue data object group | |
|---|---|---|
| bidcycm | nqcyctm | (cycle time) |
| bidloadun | nqloadun | (Load/ unload time) |
| bidmachtyp | nqmachty | (Machine type) |
| bidpartno | nqnopart | (Number of parts) |
| bidsetupm | nqsetup | (Set up time) |
| bidtooltyp | nqtoolrq | (Tools required) |
| bidjob | v#job | (Job name) |
| bidiffile | v#iffile | (IF File name) |

### 2. Bidding data object mapping from cell controllers via the view screen.

This group transports bidding information between cell controllers via the view screen. The view screen transports the information contained within the appropriate Bidding data object group at the factory controller to the same Bidding data object group at the cell controller. This is done in the process of broadcasting the Bidding objects so as to enable the cell controllers to formulate a bid from the description of the task being bid (Bidding data objects).

| Bid Information | Block name | Description |
|---|---|---|

| | | |
|---|---|---|
| Bid acknoldge | bidacklodg | (Used by cell to confirm the placing of a bid) |
| Bid cycle time | bidcyctm | (Estimated cycle time of job currently auctioned) |
| Bid load/unload time | bidloadun | (Estimate load unload time of task) |
| Bid machine type | bidmachtyp | (Machine type required to manufacture task) |
| Bid Number of parts | bidpartno | (Number of parts required to manufacture) |
| Bid set up time | bidsetupm | (Estimated extra set up time required for task) |
| Bid tools type | bidtooltyp | (Tools required to manufacture task) |
| Bid Winning cell | bidwincell | (Number of the cell to win the auction) |
| Bid Request | bidrequest | (Request for auction) |
| Bid Information file | bidiffile | (Name of information file for current job) |
| Bid Job name | bidjob | (Name of current job) |

### 3. Bidding Data object mapping from cell controller via Bidrotation application

This group transports bidding objects from the cell initiating the BidRequest to the factory controller. This mapping is identical to the first except the direction of the information flow is from the cell controller to the factory controller.

| Bid Information | Block name | Description |
|---|---|---|
| Bid acknoldge | bidacklodg | (Used by cell to confirm the placing of a bid) |
| Bid cycle time | bidcyctm | (Estimated cycle time of job currently auctioned) |
| Bid load/unload time | bidloadun | (Estimate load unload time of task) |
| Bid machine type | bidmachtyp | (Machine type required to manufacture task) |
| Bid Number of parts | bidpartno | (Number of parts required to manufacture) |
| Bid set up time | bidsetupm | (Estimated extra set up time required for task) |
| Bid tools type | bidtooltyp | (Tools required to manufacture task) |
| Bid Winning cell | bidwincell | (Number of the cell to win the auction) |
| Bid Request | bidrequest | (Request for auction) |
| Bid Information file | bidiffile | (Name of information file for current job) |
| Bid Job name | bidjob | (Name of current job) |

### 4. Bidding token data object mapping.

This group transports information about the location of the bidding token in the factory and the also about the status of the bidding rotation program. This data object group transfers information about the current entity within the factory who is able to initiate an auction. The information is transferred from a data structure within the visual basic application to the Data base blocks within the cell controller.

| Bid rotation application | Cell controller |
|---|---|

| | | |
|---|---|---|
| BiddingToken | bidtoken | (This is the bidding token) |
| Biddingoperate | bidoperate | (Bidding in operation) |
| Biddingapplicationenable | bidenable | (Bidding application is enabled) |

## C. Queuing Information Data Object Groups

This group transports Bidding information between a cell controller, it's queue control application and the other bidding entities. These data objects are the interfaces for the queue control applications and the queue database blocks within the cell controller. Information can be obtained or placed into the queue control application by first placing the information into the queue database blocks and then instructing the application to load these data objects.

### 1. Data object mapping from the Queue control application to a cell controller.

These data objects refer to the mapping's of the queue control application to the cell controllers queue data objects. The data objects within the queue control application refer to a data structure within the Visual Basic application and the data objects within the cell controller refer to database blocks.

| Queue Control (VB) | Cell Control/ Bidding Entity (FIXDMACS) | |
|---|---|---|
| Next Queue Data Object Group | | |
| Noofparts | nqcurcycno | (Current cycle number of total) |
| Setuptime | nqpartset | (Additional set up time for part) |
| Machinetype | nqmachtyp | (Machine type required) |
| Cycletime | nqcycdur | (Duration of each cycle) |
| Toolsrequired | nqtoolsreq | (Tools required) |
| Jobname | v#job | (Name of Job) |
| IFFilename | v#qiffile | (IF File name) |
| Queuenumber | nqqueno | (Current Queue Number) |
| Loadunloadtime | nqloadun | (Load/unload time) |
| Current Queue Data Object Group | | |
| IFFilename | v#file | (IFFile name) |
| Jobname | v#job | (Job name) |
| (These two control DDE links map control information from Fixdmacs to VB) | | |
| QueueControl | qcontrol | (Queue operations control) |
| IFControl | ifcontrol | (information File control) |

## 2. NodeRecord Data Object Group

This group transports cell layout information between the factory controller and the cell controllers. Information is first entered by the user at the factory controller in the fields mentioned below. This information is then placed into a data structure within a file which is stored upon the network server. At a latter data this file is then retrieved and the transported into the cell controller into the database blocks given below.

| Factory Control (VB) | File | CellControl (FIXDMACS) |
|---|---|---|
| Machine name | devicename | machname |
| Machine type | devicetype | machtyp |
| Machine Set up time | setuptime | machset |
| Machine down time | downtime | machdown |
| Machine service interval | serviceinterval | machserv |
| IF File name | iffiletype | filetyp |
| Communication ID | communicationid | commid |
| Communication Paremeters | communicationparameter | commpar |
| Part servicing type | partservicetype | partser |
| Feedback signal | feedbacksignal | feedsig |
| Communications port | communicationsport | commport |

## 3. NodelistRecord Data Object Group

This group stores Job date information, for use by the factory controller. This is the data object which stores information at the factory controller regarding the data at which a job is required to be released into the system. Information about the name of the job file (IFFile) and the due data (data to begin processing) are entered by the user. This information is then placed into a data structure within the factory controller and used by the job release queue as a guide when to release jobs into the system to begin processing.

| Factory Control (VB) | File |
|---|---|
| File Name | filename |
| Due Date | duedate |

## 4. MachineRecord Data Object Group

This group stores machine information, for use by the factory controller. This information is used by the factory controller as a definition of the different devices available within the factory. Information about the machine name is entered by the user at the factory controller, this information is then placed into a data structure and then into a file where it is used by the factory controller as a definition of the what devices are available within the factory.

| Factory Control (VB) | File |
|---|---|
| Machine name | machinename |
| Machine index | currentindex |

5.5.2.5. Data collection from FIX DMACS

The collection of data by various Visual Basic applications so as to provide summary information to the factory operator, requires the factory application to interface with the cell controller VMD. Thus the interface between the factory controller and the Visual Basic application is partially defined at one end. However the data structure at the factory controller (VB application) has yet to be defined as the high level information management and presentation functions of the factory controller have yet to be defined.

# Appendix B

## List of FIX DMACS Database Blocks for the Cell Controller

This appendix contains a list of all database blocks that a cell controller consists of, including both VMD and Bidding blocks.

## List of FIX DMACS Database Blocks

| Block Name | Type | Block Description |
|---|---|---|
| V1LOADCOMP | AI | VMD1 SO LOADDEVICE_COMPLEATED |
| V1MACHBUSY | AI | VMD 1 SO MACHINE_BUSY |
| V1MACHERR | AI | VMD 1 SO MACHINE_ERROR1 |
| V1MACHIDLE | AI | VMD 1 SO MACHINE_IDLE |
| V1MACHRUN | AI | VMD 1 SO MACHINE_RUNNING |
| V1MACHSTOP | AI | VMD 1 SO MACHINE_STOPPED |
| V1PROGREDY | AI | VMD 1 SO PROGRAM_READY |
| V1UNLOADCM | AI | VMD1 SO UNLOAD_COMPLEATE |
| BIDACKLODG | AO | BIDDING BO BID_LODGE_ACKNOLDGE |
| BIDCURQUE | AO | BIDDING PG Current Que Time |
| BIDCURTOOL | AO | BIDDING PG Current machine tool |
| BIDCYCTM | AO | BIDDING BO CYCLE_TIME |
| BIDENABLE | AO | BIDDING BO BID_ENABLE |
| BIDLOADUN | AO | BIDDING BO LOAD/UNLOAD_TIME |
| BIDLODGE | AO | BIDDING BO BID_LODGE |
| BIDMACHTYP | AO | BIDDING BO MACHINE_TYPE |
| BIDOPERATE | AO | BIDDING BO BID_OPERATE |
| BIDPARTNO | AO | BIDDING BO No_PARTS_REQUIRED |
| BIDREQUEST | AO | BIDDING BO BID_REQUEST |
| BIDSETUPM | AO | BIDDING BO SETUP_TIME |
| BIDTOKEN | AO | BIDDING BO BIDDING_TOKEN |
| BIDTOOLREQ | AO | BIDDING BO Tool Request |
| BIDTOOLTYP | AO | BIDDING BO TOOL_REQUIRED |
| BIDTRANREQ | AO | BIDDING BO Transport Request |
| BIDWINCELL | AO | BIDDING BO WINNING_CELL |
| V1BIDCYCTM | AO | VMD1 PG Start value, recalc queue time |
| V1COMMID | AO | VMD1 SO COMMUNICATION_IDPARAMETER |
| V1COMMPORT | AO | VMD1 SO COMMUNICATION_PORT |
| V1CURCYCNO | AO | VMD1 SO CURRENT_CYCNo |
| V1CYCCOMPL | AO | VMD 1 SO CYCLE_COMPLEATETIME |
| V1CYCDUR | AO | VMD 1 SO CYCLE_DURATIONTIME |
| V1CYCNO | AO | VMD1 SO No_OFCYCLES |
| V1CYCSTART | AO | VMD 1 SO CYCLE_STARTTIME |
| V1CYCTMCL2 | AO | VMD1 PG New Calculated value Cycle time |
| V1EMERSTOP | AO | VMD1 CO EMERGENCY_STOP |
| V1FILETYP | AO | VMD1 SO FILE_TYPEREQUIRED |
| V1FINALTSK | AO | VMD1 SO FINAL_TASK |
| V1IFCONTRL | AO | VMD 1 PG Information File Control |
| V1LOAD | AO | VMD 1 CO LOAD_DEVICE |

| V1LOADUNTM | AO | VMD1 SO LOAD_UNLOADTIME |
|---|---|---|
| V1MACHDOWN | AO | VMD 1 SO MACHINE_DOWNTIME |
| V1MACHINIT | AO | VMD1 CO INITIALISE_DEVICE |
| V1MACHSERV | AO | VMD1 SO MACHINE_SERVICEINTERVAL |
| V1MACHSET | AO | VMD 1 SO MACHINE_SETUPTIME |
| V1MACHTYP | AO | VMD 1 SO MACHINE_TYPE |
| V1NEWQUETM | AO | VMD1 PG New calc of v1 que time |
| V1NQCYCTM | AO | VMD1 NEXTQUEUE CYCLE_TIME |
| V1NQLOADUN | AO | VMD1 NEXTQUEUE LOAD/UNLOAD TIME |
| V1NQMACHTY | AO | VMD1 NEXTQUEUE MACHINE_TYPE |
| V1NQNOPART | AO | VMD1 NEXTQUEUE NOOF_PARTS |
| V1NQSETUP | AO | VMD1 NEXTQUEUE PARTSETUP_TIME |
| V1NQTOOLRQ | AO | VMD1 NEXTQUEUE TOOLS_REQUIRED |
| V1NXTQUENO | AO | VMD1 SO NEXT_QUEUENO |
| V1PARTSER | AO | VMD1 SO PART_SERVICINGMETHOD |
| V1PARTSET | AO | VMD1 SO PART_SETUPTIME |
| V1PROGDOWN | AO | VMD 1 CO PROGRAM_DOWNLOAD |
| V1PROGUP | AO | VMD 1 CO PROGRAM_UPLOAD |
| V1QCONTROL | AO | VMD1 CO QUEUE CONTROL |
| V1QUENO | AO | VMD1 SO QUEUE_NUMBERIN |
| V1QUETIME | AO | VMD 1 SO QUEUE_TIME |
| V1QUETOP | AO | VMD1 SO QUEUE_TOPOF |
| V1START | AO | VMD 1 CO START_DEVICE |
| V1STOP | AO | VMD 1 CO STOP_DEVICE |
| V1TOOLAVL | AO | VMD 1 SO TOOLS_AVAILABLE |
| V1UNLOAD | AO | VMD 1 CO UNLOAD_DEVICE |
| BIDCALVAL | CA | BIDDING PG Calculation of Bid |
| V1CYCTMCAL | CA | VMD1 PG New cycle time calculation |
| V1QUERECAL | CA | VMD1 PG recalc latest que time with BO |
| PGCYCTIME | DI | PG Cycle time start stop |
| PGCYCTIME2 | DI | PG Estimated cycle timer start and stop |
| V1CYCTIME2 | DO | VMD1 PG Cycle timer reset block |
| BIDCALC | PG | BIDDING PG Calculation Control |
| BIDCALCRST | PG | BIDDING PG Reset all BO |
| BIDOUTCON | PG | BIDDING PG Outgoing bidding control |
| BIDPROG | PG | BIDDING PG Starting of Bidding Calc |
| V1CONTROL | PG | VMD1 PG Master Control Block |
| V1DWNCON | PG | VMD1 PG Set down Control |
| V1IDLECON | PG | VMD1 PG Idle wait sate control |
| V1IDLERST | PG | VMD1 PG Idle reset all AI |
| V1PRTREBID | PG | VMD1 BIDDING PG Rebid rest of job |
| V1REBIDCON | PG | VMD1 PG Rebid next task from nqueue |
| V1RUNCON | PG | VMD1 PG Run & Monitor Control |
| V1RUNCON2 | PG | VMD1 PG Run & Monitor (Timing) |
| V1RUNCON3 | PG | VMD1 PG Running & Monitoring (Timing) |
| V1SETCON | PG | VMD1 PG Setup device control |
| V1WINBID | PG | VMD1 PG Winning of bid control |

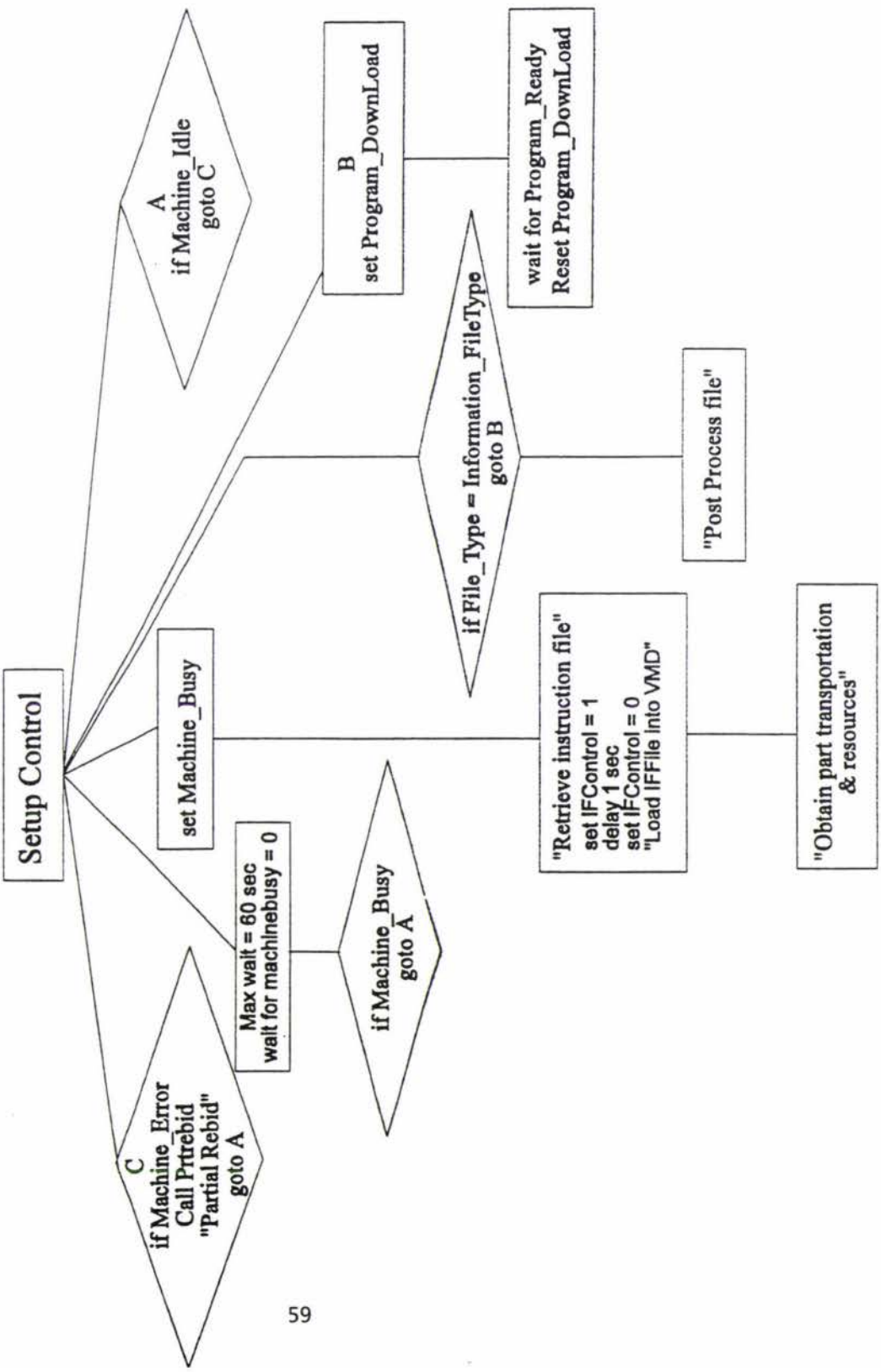
| | | |
|---|---|---|
| V1CYCTIME | TM | VMD1 SO CYCLE_TIMEOFCURRENT |
| V1ESTCYCTM | TM | VMD1 SO ESTMATED_CYCLETIME |
| BIDIFFILE | TX | ---- |
| BIDJOB | TX | ---- |
| V1COMMPAR | TX | ---- |
| V1FEEDSIG | TX | ---- |
| V1FIXREQ | TX | ---- |
| V1GEOFILE | TX | ---- |
| V1IFFILE | TX | ---- |
| V1JOBNAME | TX | ---- |
| V1MACHFILE | TX | ---- |
| V1MACHNAME | TX | ---- |
| V1NQIFFILE | TX | ---- |
| V1NQJOB | TX | ---- |
| V1TOOLSREQ | TX | ---- |

# Appendix C

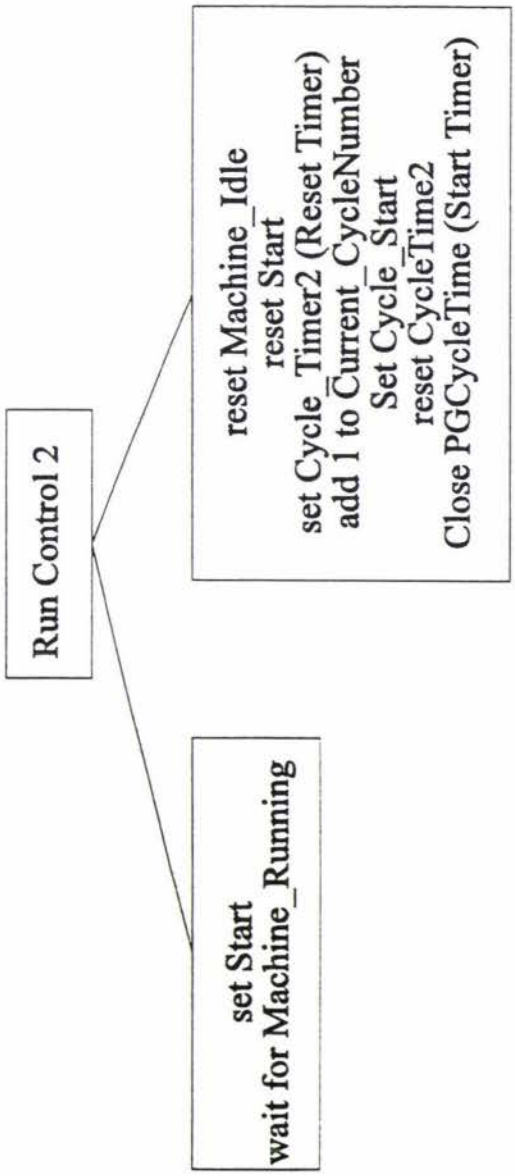
## Diagrams for the Device Control Algorithm Program Blocks

This appendix contains a series of diagrams which explain the functionality of each program block that the DCA consists of. Also contained within this appendix is a description of the calculation blocks used within the DCA.

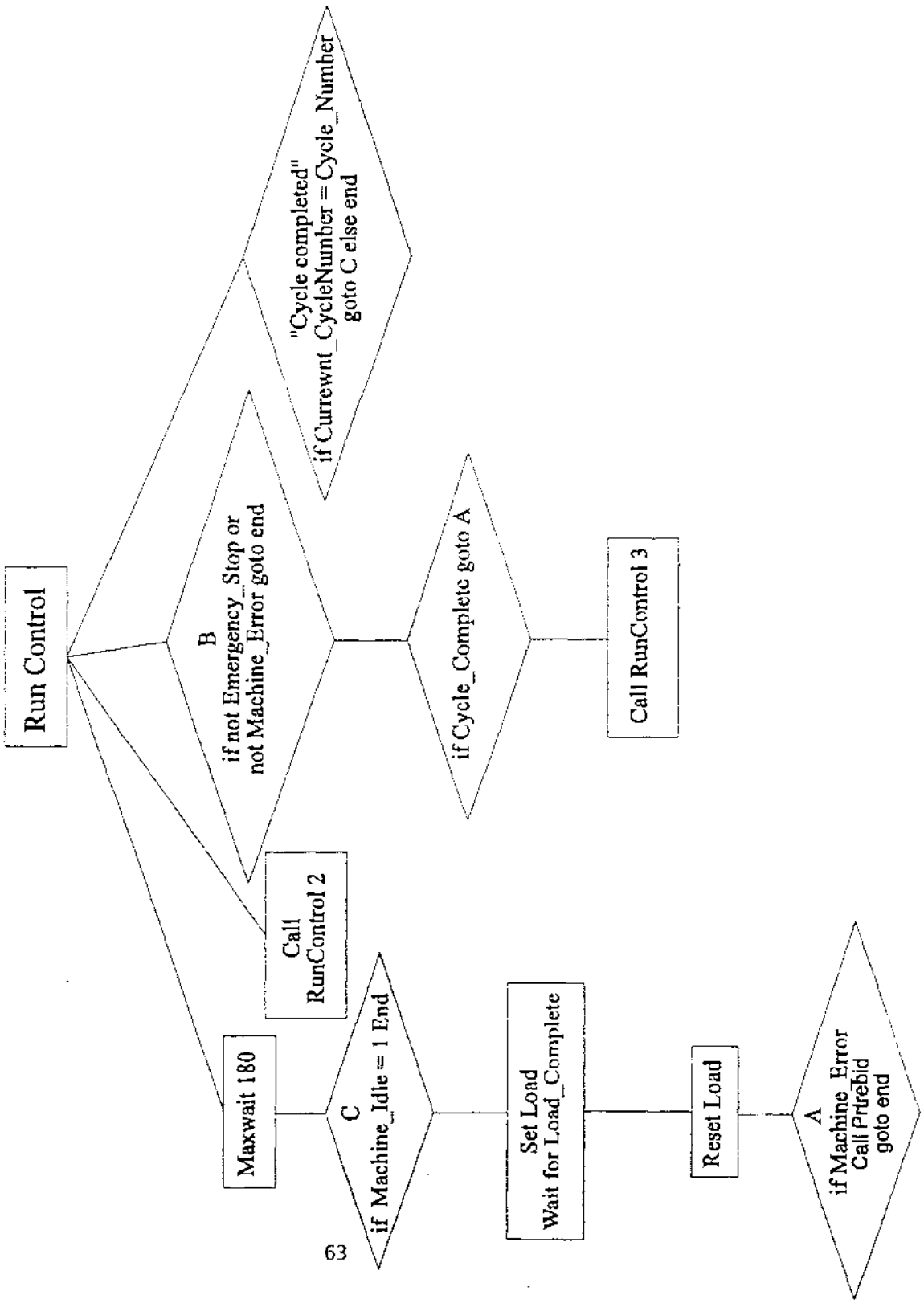
## Functional diagram of Set up Control Program Block



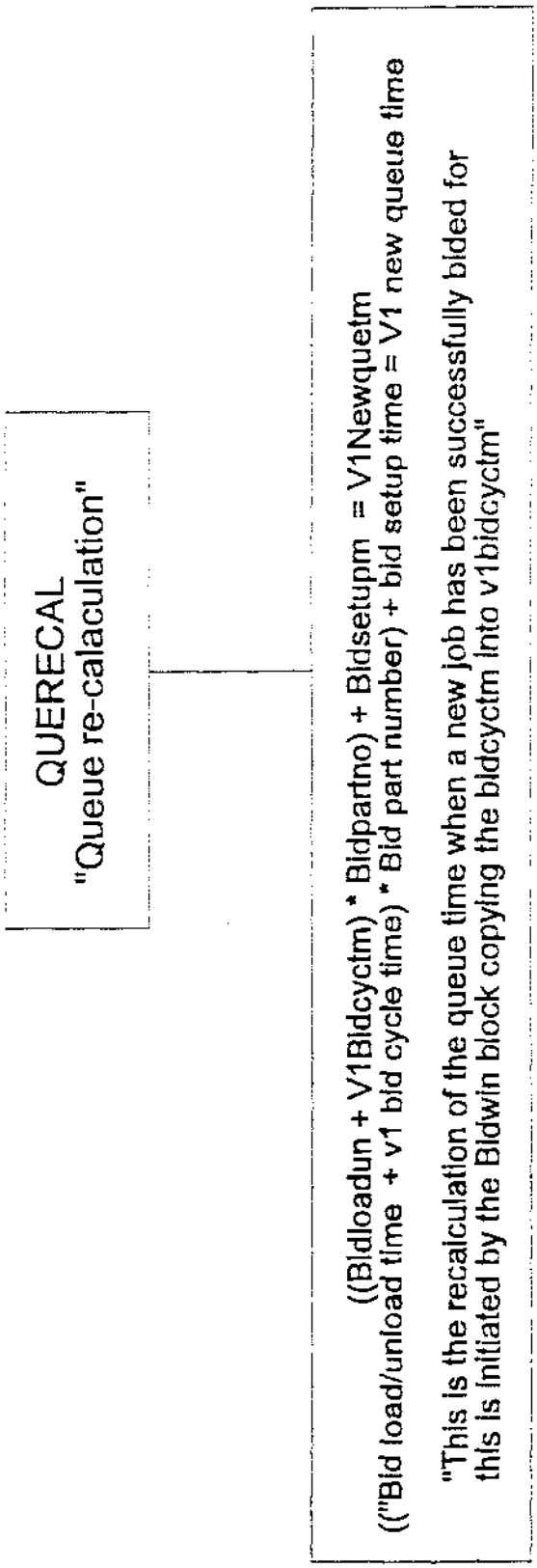
## Functional diagram of Run Control 2 Program Block



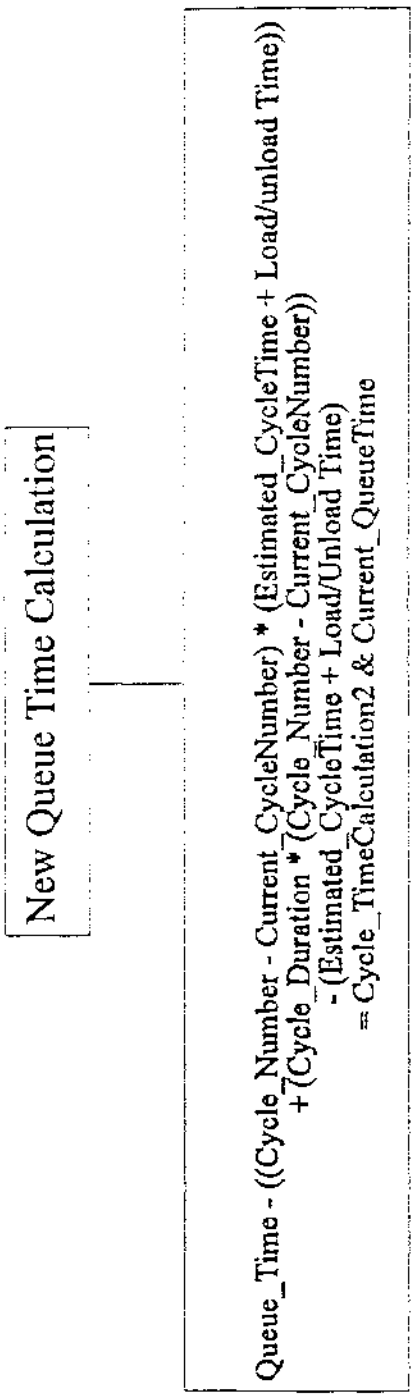
# Functional diagram of Run Control Program Block



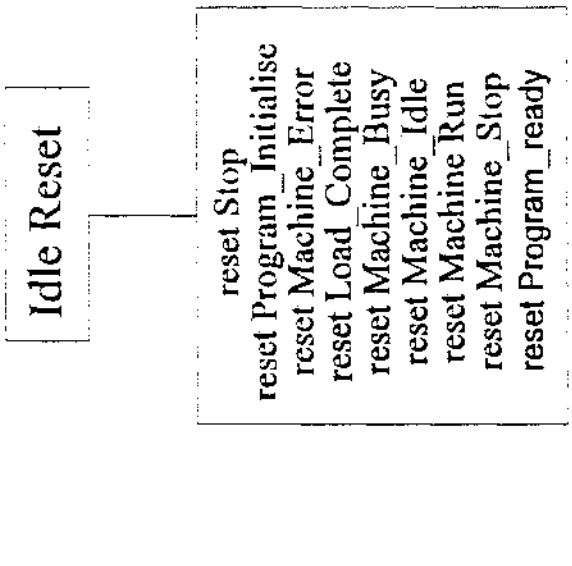
# Diagram of Queue Re-calculation Calculation Block



## Diagram of New Queue Time Calculation Calculation Block



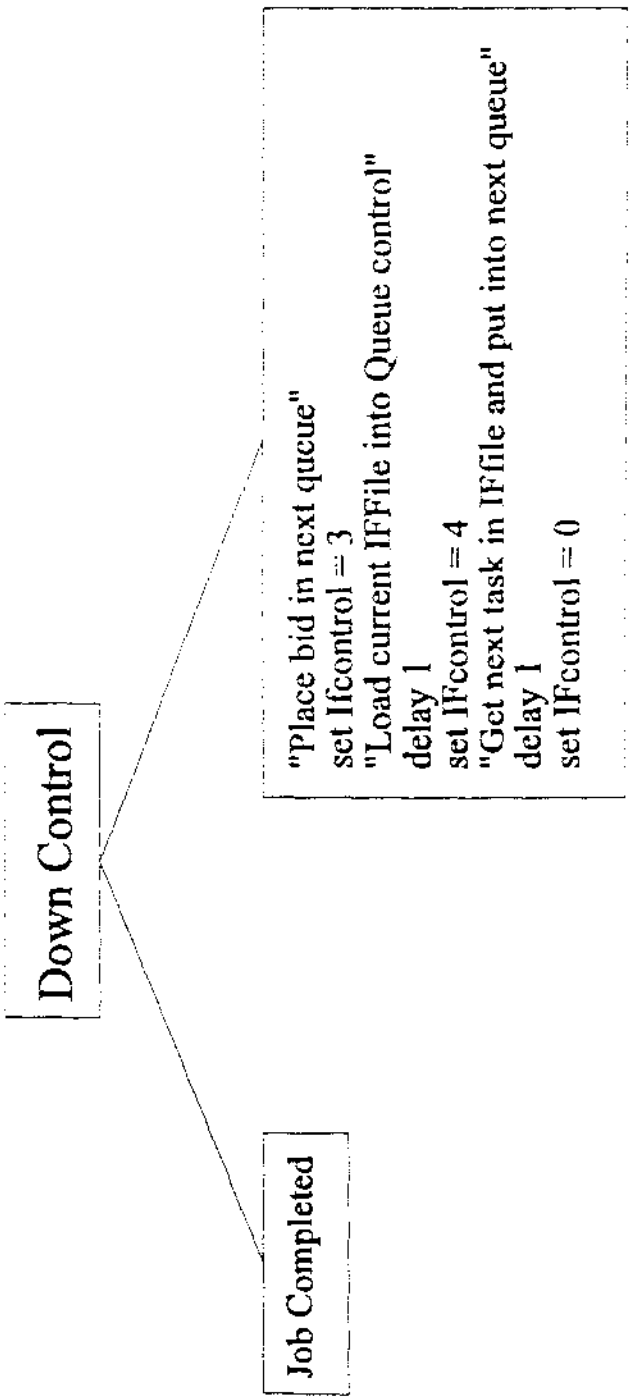
## Functional diagram of Idle Reset Program Block



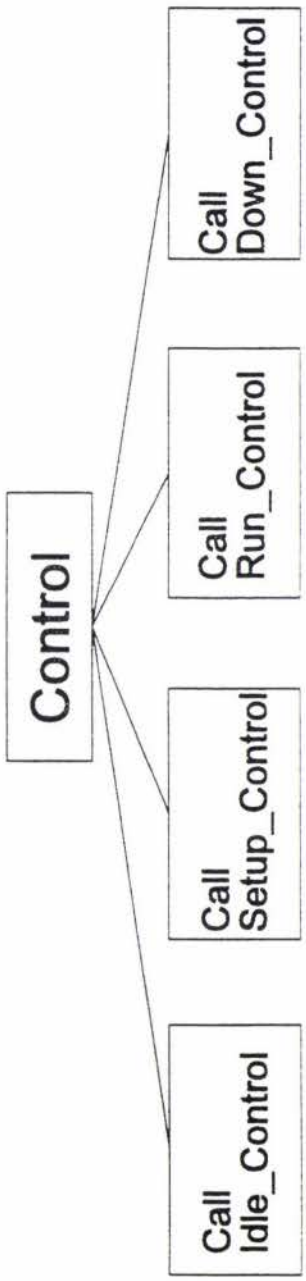
(Queue_Number = Number In Current Queue)

Functional diagram of Idle Control Program Block

Functional diagram of Down Control Program Block



## Functional diagram of Master Control Program Block



# Appendix D

## Diagrams for the Auction/Bidding Algorithms

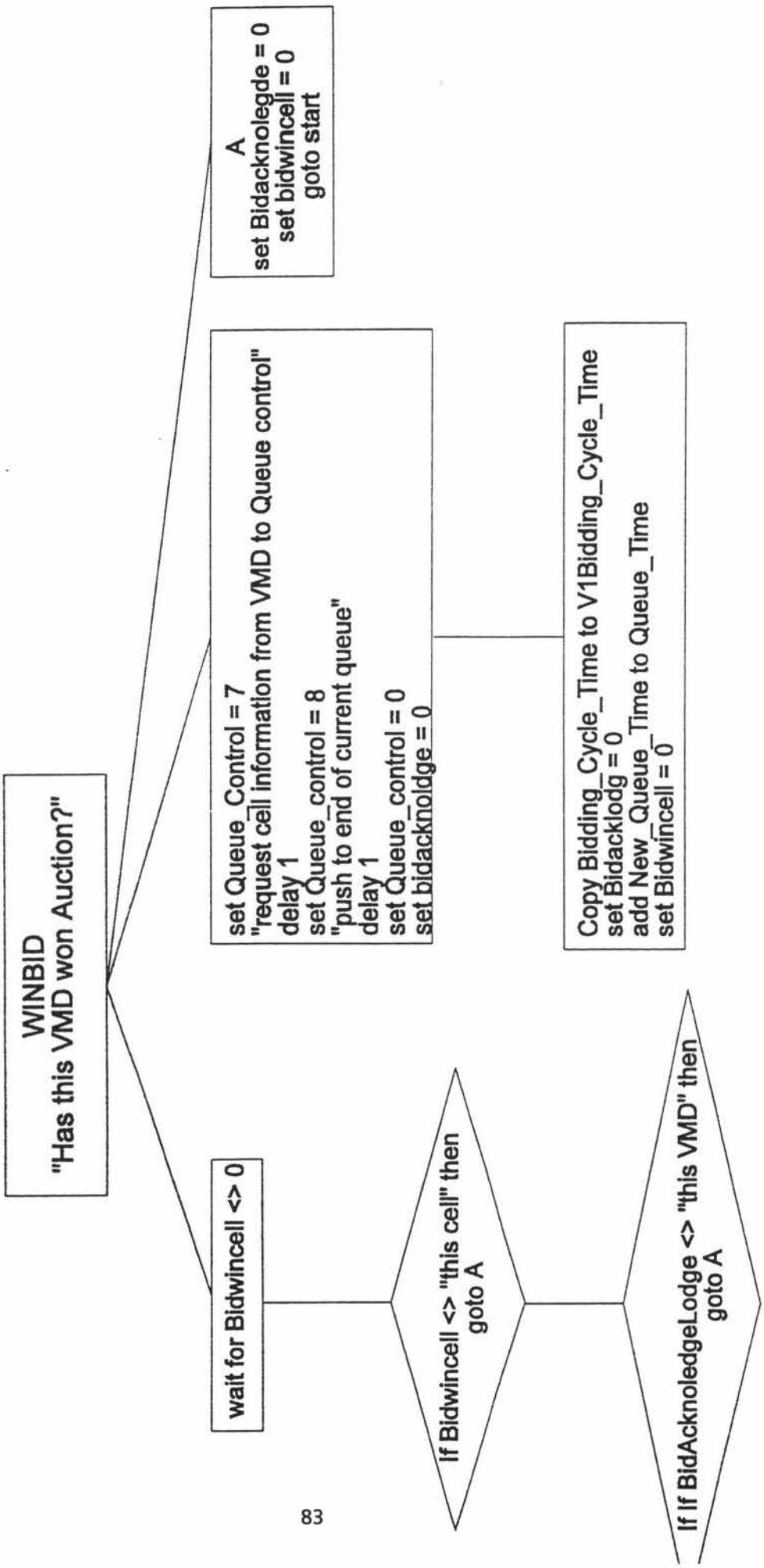
This appendix consists of a series of diagrammatic representations of the program and calculation blocks which make up the Auction or Bidding algorithm.

## Variable definitions for Bidding Algorithms

| Variable Name | Explanation |
|---|---|
| *BidChoose* | |
| BidRequest | Request Set by Factory controller for a bid to be lodged. |
| BidBegwait | Starts timer for a preset period of time, if no bids are lodged after a preset waiting time then the program is reset. |
| Bidsetwait | Preset timer is finished and program is reset. |
| BidAckC1 or BidAcknowledge | Bid Acknowledge Cell 1, cell 1 has acknowledged that its bid has been lodged. |
| *BidChoose1* | |
| BidlodgedCell1 or BidLogC1 | This contains the Bid Cell 1 has lodged in response to the request. |
| *BidCalc* | |
| BidMachtyp | Machine type required to manufacture the part which originated the bidding request. |
| BidCurque | The queue wait time for machine in cell able to manufacture the part. |
| BidCurtool | The tool available for the machine in the cell able to manufacture the part. |
| V1Quetime | The queue wait time for the machine 1 in the cell. |
| V1ToolAvl | The tool available to machine 1 in the cell. |
| Bidtooltyp | The tool required to manufacture the part which originated the bidding request. |
| BidtoolReq | Request of tool required for the machine in the cell to produce the part. |
| BidtransReq | Request for transportation for the part to be operated on for the machine in the cell which will undertake this operation. |
| *BidCalcVal* | |

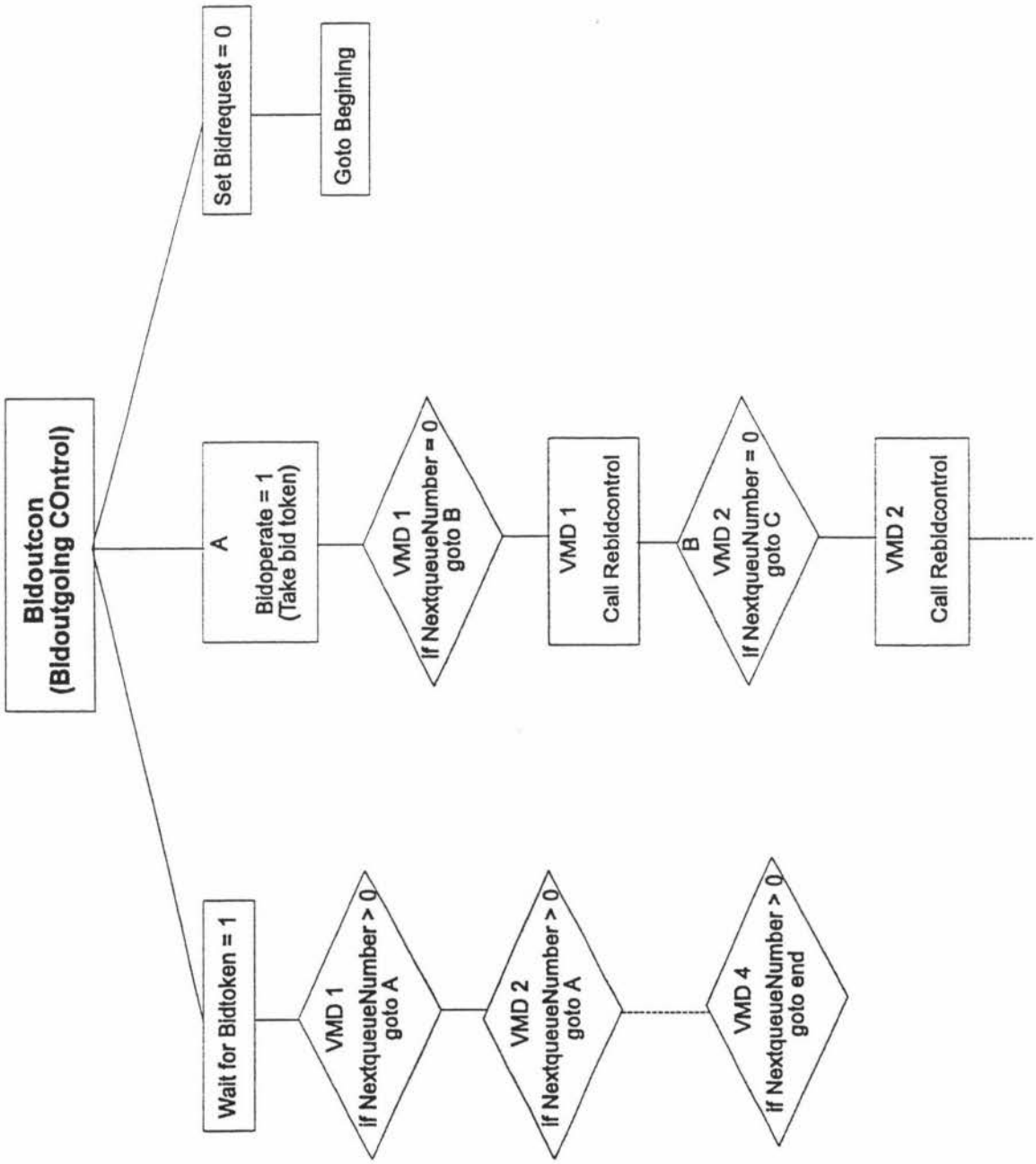
| | |
|---|---|
| BidPartNo | Number of parts in order to satisfy the bid request to be manufactured. |
| Bidcyctm | Cycle time of the part to be manufactured. |
| Bidsetuptm | Set up time of the part to be produced. |

## Functional diagram of Winning Bid Program Block

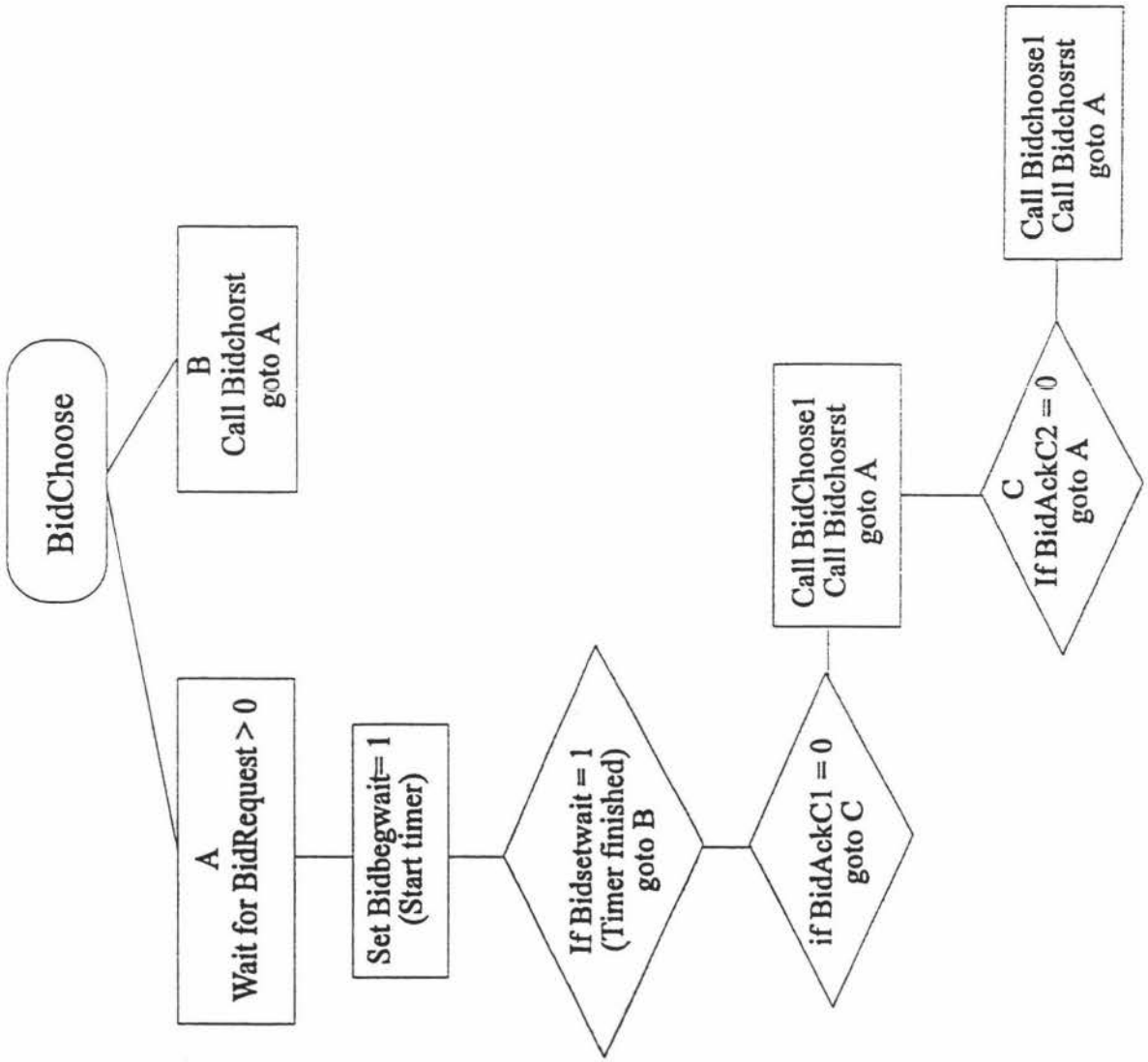


**Functional diagram of Master Bidding Algorithm Program Block**

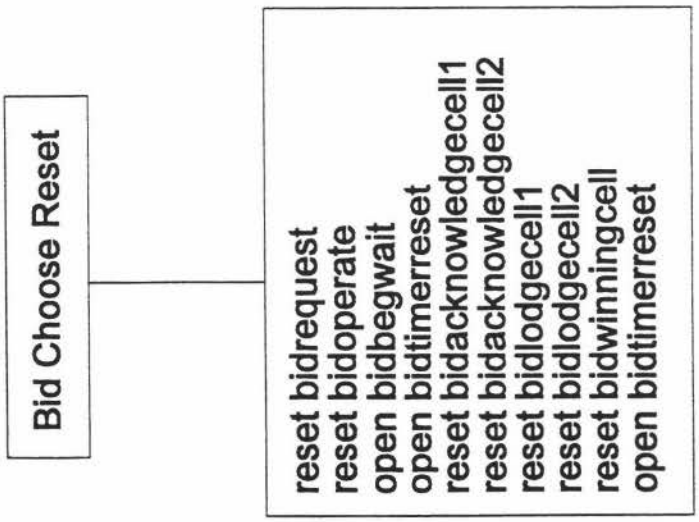
# Functional diagram of Outgoing Bid Control Program Block



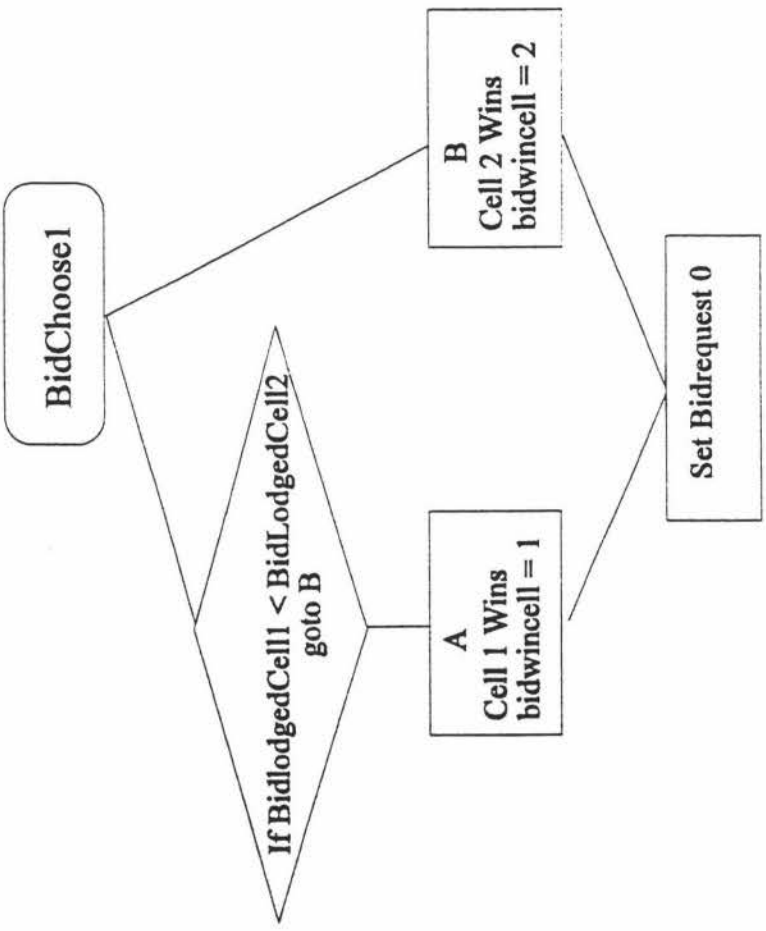
## Functional diagram of Bid Choose Program Block



## Functional diagram of Bid Choose Reset Program Block

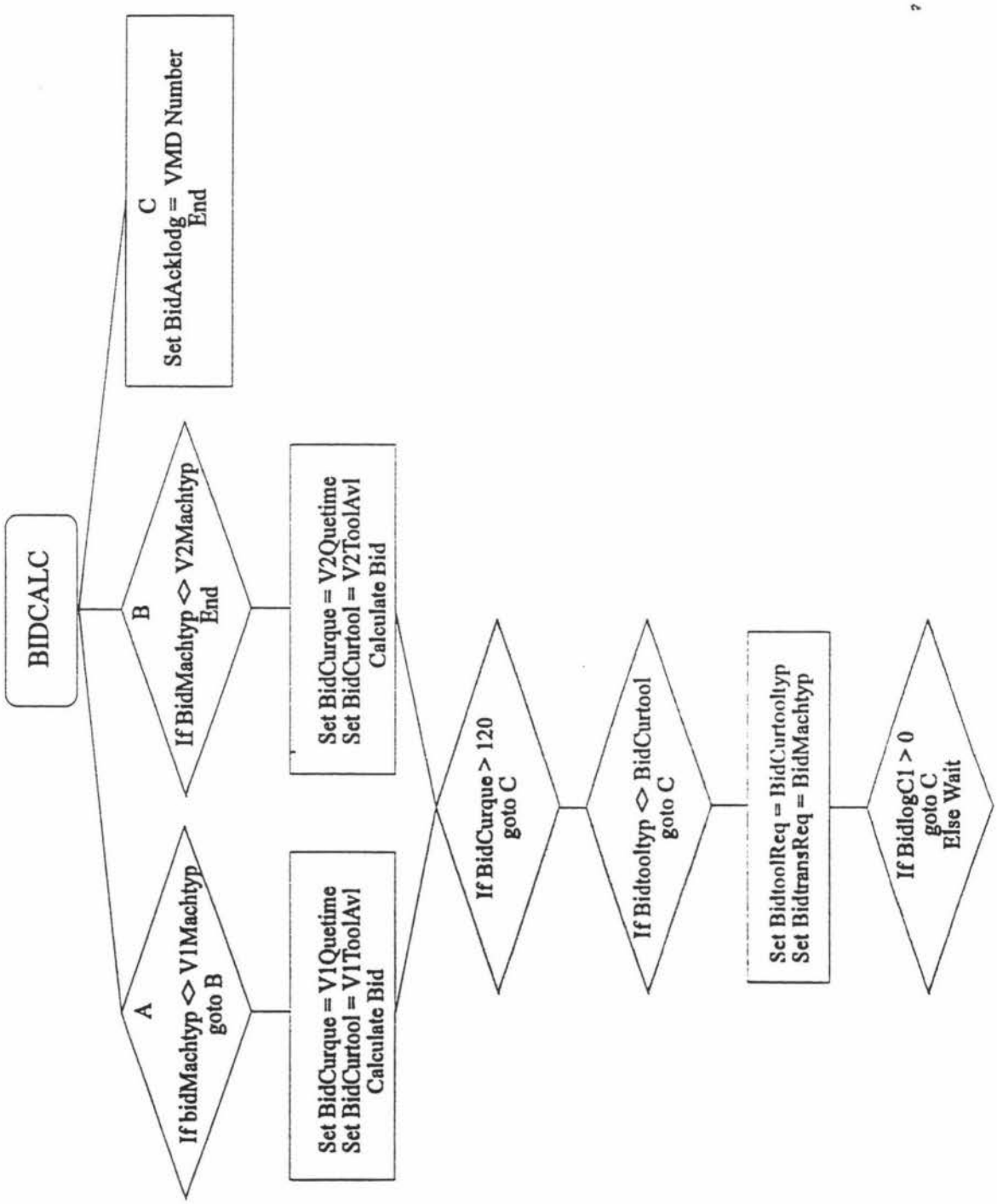


## Functional diagram of Bid Choose 1 Program Block

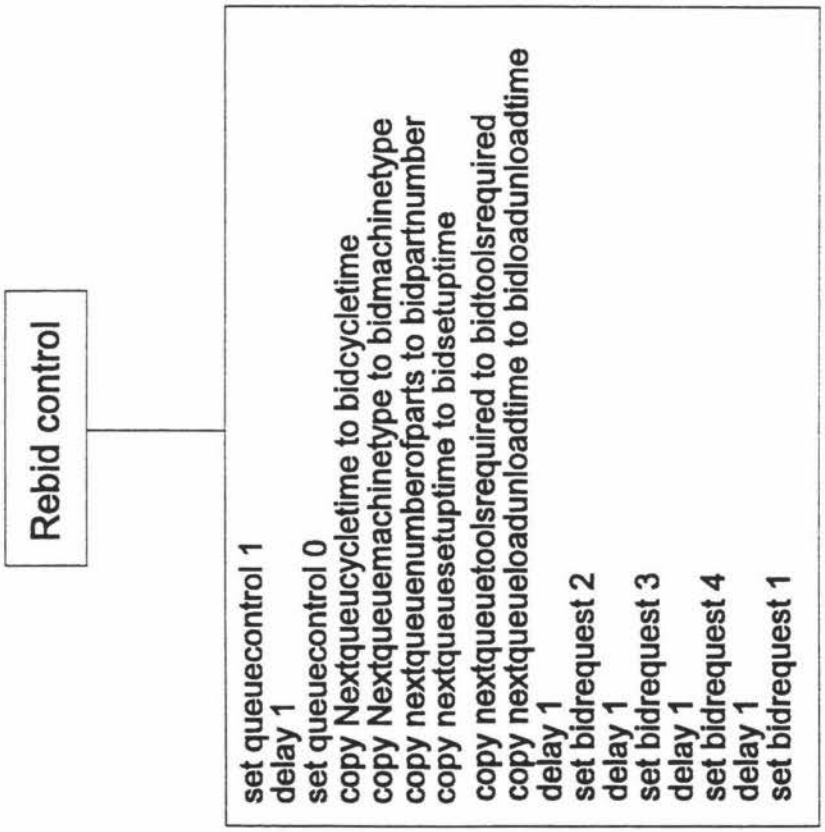


Functional diagram of Partial Re-bid Control Program Block

## Functional diagram of Bid Calculation Control Program Block



## Functional diagram of Re-bid Control Program Block



# Appendix E

## List of Data Objects and Data Object Mapping's

This appendix consists of a list of all data objects and their respective mapping's, used to transfer information between various entities both within and outside FIX DMACS.

# DATA OBJECT MAPS

This section shows the mapping of grouped data objects between different information entities. Often several groups map from different address to the same destination. The word transport has been used interchangeably with mapping, as they both refer to the transference of information from one place to another. Their are three ways in which this is accomplished. 1. DDE transfers, this transports information via Dynamic Data Exchange between FIXDMACS and a Visual Basic application. 2. Transfers within a FIXDMACS node or cell, this is achieved by coping data from one block to another. 3. Transfer between FIXDMACS nodes via the view screen, this uses FIXDMACS own programming lanugo to transfer information between two data base block.

## JobRecord Data Object Group

This group transports Job information between the factory controller and the cell controllers, this occurs via file.

| **Factory Control (VB)** | **- File -** | **Cell Control (FIXDMACS)** |
|---|---|---|
| Name of Job | Jobname | Jobname |
| Estimated cycle time | Estcycletime | estcyctm |
| Setup time | Setuptime | Partset |
| Machine data file name | Machinefilename | Machfile |
| Geometric file name | Geometricfilename | geofile |
| Fixtures information | Fixturesfilename | fixreq |
| Tooling Information | Toolsrequired | toolsreq |
| Final Task | Finaltask | finaltask |
| Number of cycles | Noofcycles | cycno |
| Estimate load/unload time | Estloadunloadtime | loaduntm |

(These objects are used for the control of this transfer)

| | |
|---|---|
| (Task index number) | Currentindex |
| (Total number of tasks) | Tasknumber |
| Task Name | Taskname |
| (Release control) | Duedate |
| (Queue force control) | Force |
| (Job operation control) | Status |

## Bidding Information Data Object Groups

### 1. Bidding data object group maps

This group maps bidding information between the cell controllers bidding entity and the cell controllers next queue data object.

| Bidding entity data object group | Next queue data object group | |
|---|---|---|
| bidcycm | nqcyctm | (cycle time) |
| bidloadun | nqloadun | (Load/ unload time) |
| bidmachtyp | nqmachty | (Machine type) |
| bidpartno | nqnopart | (Number of parts) |
| bidsetupm | nqsetup | (Setup time) |
| bidtooltyp | nqtoolrq | (Tools required) |
| bidjob | v#job | (Job name) |
| bidiffile | v#iffile | (IF File name) |

### 2. Bidding data object mapping from cell controllers via the view screen.

This group transports bidding information between cell controllers via the view screen.

**Factory controller to cell controller links**

| | | |
|---|---|---|
| Bid acknoldge | bidacklodg | (Used by cell to confirm the placing of a bid) |
| Bid cycle time | bidcyctm | (Estimated cycle time of job currently auctioned) |
| Bid load/unload time | bidloadun | (Estimate load unload time of task) |
| Bid machine type | bidmachtyp | (Machine type required to manufacture task) |
| Bid Number of parts | bidpartno | (Number of parts required to manufacture) |
| Bid setup time | bidsetupm | (Estimated extra setup time required for task) |
| Bid tools type | bidtooltyp | (Tools required to manufacture task) |
| Bid Winning cell | bidwincell | (Number of the cell to win the auction) |
| Bid Request | bidrequest | (Request for auction) |
| Bid Information file | bidiffile | (Name of information file for current job) |
| Bid Job name | bidjob | (Name of current job) |

### 3. Bidding Data object mapping from cell controller via Bidrotation application

This group transports bidding objects from the cell initiating the BidRequest to the factory controller.

**Cell controller to factory controller links**

| | | |
|---|---|---|
| Bid acknoldge | bidacklodg | (Used by cell to confirm the placing of a bid) |
| Bid cycle time auctioned) | bidcyctm | (Estimated cycle time of job currently |
| Bid load/unload time | bidloadun | (Estimate load unload time of task) |
| Bid machine type | bidmachtyp | (Machine type required to manufacture task) |
| Bid Number of parts | bidpartno | (Number of parts required to manufacture) |
| Bid setup time | bidsetupm | (Estimated extra setup time required for task) |
| Bid tools type | bidtooltyp | (Tools required to manufacture task) |
| Bid Winning cell | bidwincell | (Number of the cell to win the auction) |
| Bid Request | bidrequest | (Request for auction) |
| Bid Information file | bidiffile | (Name of information file for current job) |
| Bid Job name | bidjob | (Name of current job) |

### 4. Bidding token data object mapping.

This group transports information about the location of the bidding token in the factory and the also about the status of the bidding rotation program.

| **Bid rotation application** | **Cell controller** | |
|---|---|---|
| (These links are between the bidding rotation control application and the cell with the bidding token) | | |
| BiddingToken token) | bidtoken | (This is the bidding |
| Biddingoperate | bidoperate | (Bidding in operation) |
| (This link is between all the cells) | | |
| Biddingapplicationenable | bidenable | (Bidding application is enabled) |

## Queuing Information Data Object Group

This group transports Bidding information between a cell controller, it's queue control application and the other bidding entities.

### 1. Data object mapping from the Queue control application to a cell controller.

| Queue Control (VB) | Cell Control/ Bidding Entity (FIXDMACS) | |
|---|---|---|
| (For the Next Queue operation) | | |
| Noofparts | nqcurcycno | (Current cycle number of total) |
| Setuptime | nqpartset | (Additional setup time for part) |
| Machinetype | nqmachtyp | (Machine type required) |
| Cycletime | nqcycdur | (Duration of each cycle) |
| Toolsrequired | nqtoolsreq | (Tools required) |
| Jobname | v#job | (Name of Job) |
| IFFilename | v#qiffile | (IF File name) |
| Queuenumber | nqqueno | (Current Queue Number) |
| Loadunloadtime | nqloadun | (Load/unload time) |
| (For the Current Queue operation) | | |
| IFFilename | v#file | (IFFile name) |
| Jobname | v#job | (Job name) |
| (These two control DDE links map control information from Fixdmacs to VB) | | |
| QueueControl | qcontrol | (Queue operations control) |
| IFControl | | |

### 2. NodeRecord Data Object Group

This group transports cell layout information between the factory controller and the cell controllers.

| Factory Control (VB) | - File - | Cell Control (FIXDMACS) |
|---|---|---|
| Machine name | devicename | machname |
| Machine type | devicetype | machtyp |
| Machine Setup time | setuptime | machset |
| Machine down time | downtime | machdown |
| Machine service interval | serviceinterval | machserv |
| IF File name | iffiletype | filetyp |
| Communication ID | communicationid | commid |
| Communication Paremeters | communicationparameter | commpar |
| Part servicing type | partservicetype | partser |
| Feedback signal | feedbacksignal | feedsig |
| Communications port | communicationsport | commport |

### 3. NodelistRecord Data Object Group

This group stores Job date information, for use by the factory controller.

| **Factory Control (VB)** | **- File** |
|---|---|
| File Name | filename |
| Due Date | duedate |

### 4. MachineRecord Data Object Group

This group stores machine information, for use by the factory controller.

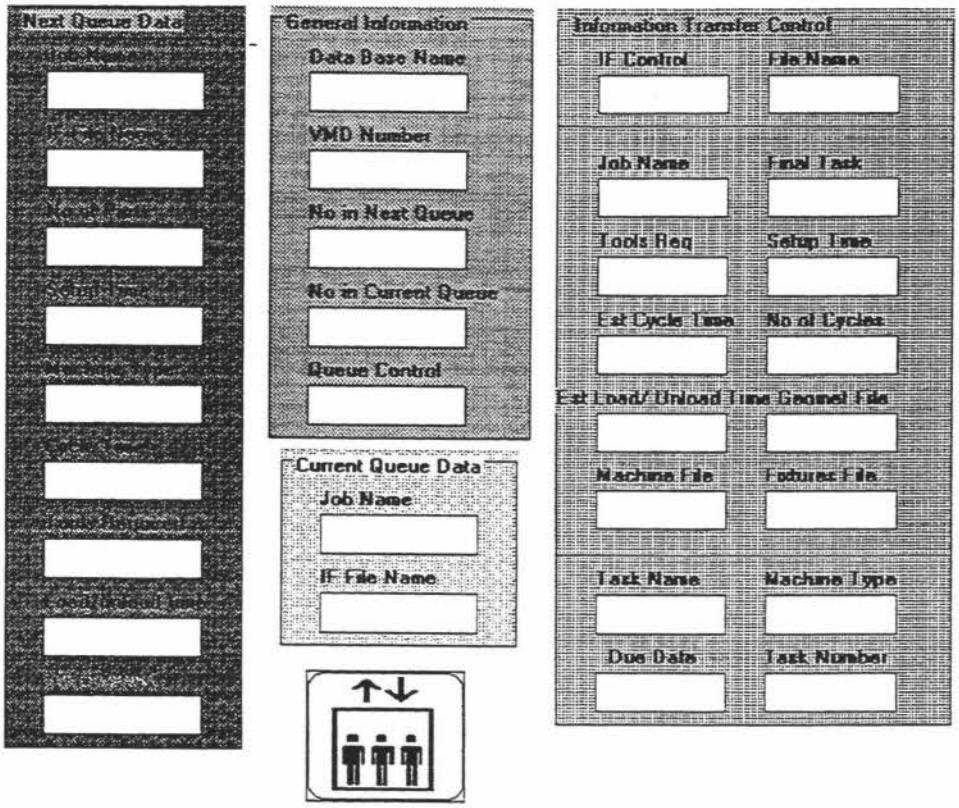
| **Factory Control (VB)** | **- File** |
|---|---|
| Machine name | machinename |
| Machine index | currentindex |

# Appendix F

## Program Listing for Next Queue Visual Basic Program

This appendix contains a complete program listing of all windows and respective program code with which the Next Queue application consists of. The listing of this application is divided up in to a series of sections, each beginning with a print out of a user screen (window) which the user can view and interact with. Following the print out of the window is a complete listing of all the code associated with the window. This listing continues until the next window is shown.

# Next Queue application main window



```
NEXTQUE.FRM - 1

Option Explicit

Option Base 1                'Begins the arrays at 1

Dim Nextqueue(100) As queue
Dim Currentqueue(100) As queue

Const Close_DDE = 5, Open_DDE = 1, NONE = 0, Link_Manual = 2, Link_Automatic = 1
  'Constants for 1 and 5

'Define factory controller
Const Factory_Controller = "cim2"

Dim Nstartofqueue As Integer
Dim NEndofqueue As Integer
Dim Cstartofqueue As Integer
Dim CEndofqueue As Integer

'For the Pop and Put function and procedures both Job_Name and IF_Filename
'require the name of the Queue to be entered.  For the rest of the Queue name
'is set for "thisqueue".
'The Pop function returns the value specified by the name
'The Put procedure passes in the postion in the queue and the value to be inserted
' at present the position to insert into the queue can only be at the end
' or at the start of the queue.  A 0 indicates the begining of the queue whilst
' a 1 indicates the end of the queue.

Dim LinkItem As String
Dim DDETextName As Variant

Dim VMDName As String

Const Finishofqueue = 100
Const Beginofqueue = 1


Sub CommDDELink (LinkItem, DDETextName As TextBox)

'This procedure is designed to run once on startup to set up a DDE link with fixdmacs
'The Block name is passed in as Linkitem and the DDE Text box name is passed in as
' DDETextName



If DDETextName.LinkMode = NONE Then              'If not already open
        On Error Resume Next
        DDETextName.LinkTopic = "DMDDE|DATA"     'set topic
        DDETextName.LinkItem = LinkItem          'set link item, this is passed in
        DDETextName.LinkMode = Link_Manual    'link mode is automatic
        End If


On Error GoTo Errorhandel

Exit Sub

Errorhandel:
    MsgBox " An Error Has Occured in Setting up the DDE Links! "
    Exit Sub
End Sub

Sub CommDDELink2 (LinkItem, DDETextName As TextBox)
'This procedure is designed to run once on startup to set up a DDE link with fixdmacs
'The Block name is passed in as Linkitem and the DDE Text box name is passed in as
' DDETextName

On Error Resume Next
DDETextName.LinkTopic = "DMDDE|DATA"     'set topic
DDETextName.LinkItem = LinkItem          'set link item, this is passed in
DDETextName.LinkMode = Link_Manual    'link mode is automatic

End Sub

Sub Writetoscreen (record As jobrecord)

job_Name.Text = record.jobname
task_name.Text = record.taskname
```

```
NEXTQUE.FRM - 2

est_cycletime.Text = record.estcycletime
setup_time.Text = record.setuptime
Est_loadunloadtime.Text = record.estloadunloadtime
machine_type.Text = record.machinetype
No_ofcycles.Text = record.noofcycles
final_task.Text = record.finaltask
Machine_File.Text = record.machinefilename
Geometric_file.Text = record.geometricfilename
tools_required.Text = record.toolsrequired
fixtures_file.Text = record.fixturesfilename
due_date.SelText = record.duedate

End Sub

Sub Databasetext_Change ()
' This text block contains the name of the node that this queue will be communicating with

End Sub

Sub Form_Load ()
'
'
'This application is written By Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994
'
'**This application is a Queueing program
'DDE links are used to transfer information between Fix Dmacs and This application
'On text box has an automatic link, this controls all of the functioning of the program
'The other text box's contain either setup information or Data to be stored or retrieved from
 the queue


'Initialise queue pointers for the array
Nstartofqueue = Beginofqueue
NEndofqueue = Beginofqueue
Cstartofqueue = Beginofqueue
CEndofqueue = Beginofqueue


' InputBox's to enter database name and vmd number oh year oh yaa ya ya ya yaaaaa forget abou
t the last one get yourself another
Databasetext.Text = InputBox("Please enter Node Name", "Queue Definition", "IEL1")
VMDNumbertext.Text = InputBox("Please enter the VMD number for this QUEUE", "Queue definition
", "1")


'Set up the automatic link for the transfer of control information between Fix and VB
If quecontroltext.LinkMode = NONE Then              'If not already open
        On Error Resume Next
        quecontroltext.LinkTopic = "DMDDE|DATA"     'set topic
        quecontroltext.LinkItem = Databasetext.Text & "." & "V" & VMDNumbertext.Text & "qcont
rol.A_CV"           'set link item, this is passed in
        quecontroltext.LinkMode = Link_Automatic    'link mode is automatic
        End If

On Error GoTo Errorhandel2


If loadApplication("DMDDE.exe", "c:\wdmacs\") = True Then  'Loads dde server


 'Set up Manual Links for all of the text box's
 ' Next Queue

'These are the push links between the next queue and the factory controller

Call CommDDELink(Factory_Controller & ".bidcyctm.A_CV", NCycletimetext)
Call CommDDELink(Factory_Controller & ".bidloadun.A_CV", NLoadunloadtimetext)
Call CommDDELink(Factory_Controller & ".bidmachtyp.A_CV", NMachinetypetext)
Call CommDDELink(Factory_Controller & ".bidpartno.A_CV", NNoofpartstext)
Call CommDDELink(Factory_Controller & ".bidsetupm.A_CV", Nsetuptimetext)
Call CommDDELink(Factory_Controller & ".bidtooltyp.A_CV", NToolsRequiredtext)
Call CommDDELink(Factory_Controller & ".bidiffile.A_CV", NIFFilenametext)
Call CommDDELink(Factory_Controller & ".bidjob.A_CV", Njobnametext)
Call CommDDELink(Factory_Controller & ".bidrequest.A_CV", NBidrequesttext)

'This is the queue number link to the cell controller
```

NEXTQUE.FRM - 3

```
Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nxtqueno.A_CV", NoinNq
ueuetext)

 ' Current Queue
 'These links are valid only for a push controlled by queue control
 Call CommDDELink(Databasetext.Text & "." & "bidjob.A_CV", CJobnameText)
 Call CommDDELink(Databasetext.Text & "." & "bidiffile.A_CV", CIFFileNameText)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "queno.A_CV", NoinCque
uetext)



'** This part of the application controls the Information file loading and transfering betwee
n file and DDE
'
'Set up the automatic link for the transfer of control information between Fix and VB
If IF_Control.LinkMode = NONE Then              'If not already open
        On Error Resume Next
        IF_Control.LinkTopic = "DMDDE|DATA"     'set topic
        IF_Control.LinkItem = Databasetext.Text & "." & "V" & VMDNumbertext.Text & "ifcontrl.
A_CV"           'set link item, this is passed in
        IF_Control.LinkMode = Link_Automatic    'link mode is automatic
        End If

End If

 'Set up Manual Links for all of the text box's
 ' IF File transfer
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "jobname.A_CV", job_Na
me)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "estcyctm.A_CV", est_c
ycletime)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "partset.A_CV", setup_
time)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "machfile.A_CV", Machi
ne_File)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "geofile.A_CV", Geomet
ric_file)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "fixreq.A_CV", fixture
s_file)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "Toolsreq.A_CV", tools
_required)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "finaltsk.A_CV", final
_task)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "cycno.A_CV", No_ofcyc
les)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "loaduntm.A_CV", Est_l
oadunloadtime)
 Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "iffile.A_CV", File_Na
me)

Exit Sub

Errorhandel2:
    MsgBox " An Error Has Occured in Setting up the DDE Links ", MB_ICONQUESTION, "DDE Link S
tatus"
    Exit Sub

End Sub

Sub IF_Control_Change ()

' 1 This causes this program to pop the name of the top of the current queue then test it's
' status bit and load the current task from the IF file into the VMD specified.
' 3 This causes the program to get filename from VMD
' 4 This causes the program load the file specified in 3 then put to the
'Next queue

Dim recordlen As Integer
position = 1
Dim defaultpath As String

On Error Resume Next

defaultpath = "r:\fix\"

Select Case IF_Control
```

```
    Case "A" To "Z", "", "a" To "z", "!" To "/", ":" To "@", "[" To "`", "{" To "~", " " To "
                    "
        'Any other char that may be input by accident

    Case Is = 1
        'This case retrieves the file name from the current queue and pushes into VMD

        'Retreave from file
        'Pop from current queue
        quecontroltext.Text = 0
        quecontroltext.Text = 6
        File_Name.Text = CIFFileNameText.Text

        'retrieve from file into jobrecord type
        'Create new file specs
        recordlen = Len(recordname)
        filenum = FreeFile

        ' Opening file
        Open defaultpath & File_Name.Text For Random As filenum Len = recordlen

        'Read from file
        Call readfromfile(recordname)

        'Add one to status bit
        If recordname.finaltask > 0 Then
            recordname.status = 1
            Else
            recordname.status = recordname.status + 1
            Call savetofile(recordname)
            End If

        'Retrieve from flie correct information
        position = recordname.status
        Call readfromfile(recordname)

        Call Writetoscreen(recordname)

        'Poke all information from file into VMD
        job_Name.LinkPoke
        final_task.LinkPoke
        tools_required.LinkPoke
        setup_time.LinkPoke
        est_cycletime.LinkPoke
        No_ofcycles.LinkPoke
        Est_loadunloadtime.LinkPoke
        Geometric_file.LinkPoke
        Machine_File.LinkPoke
        fixtures_file.LinkPoke
        File_Name.LinkPoke


        Close filenum

    Case Is = 3
        'Get filename from VMD

        'Get filename
        File_Name.LinkRequest

    Case Is = 4
        'load the filename then test the status bit then put to the next queue
        recordlen = Len(recordname)
        filenum = FreeFile
        position = 1

        ' Opening file
        Open defaultpath & File_Name.Text For Random As filenum Len = recordlen
        'Read from file
        Call readfromfile(recordname)

        If Not recordname.finaltask = 1 Then


        'Add one to status bit
        recordname.status = recordname.status + 1

        'Retrieve from flie correct information
```

```
        position = recordname.status
        Call readfromfile(recordname)

        'Write information to screen
        Call Writetoscreen(recordname)

        'Put information from file into Next queue
        Njobnametext.Text = recordname.jobname
        NIFFilenametext.Text = File_Name.Text
        NNoofpartstext.Text = recordname.noofcycles
        Nsetuptimetext.Text = recordname.setuptime
        NMachinetypetext.Text = recordname.machinetype
        NCycletimetext.Text = recordname.estcycletime
        NToolsRequiredtext.Text = recordname.toolsrequired
        NLoadunloadtimetext.Text = recordname.estloadunloadtime

        'Then push onto next queue at the end
         quecontroltext.Text = 0
         quecontroltext.Text = 3
         Close filenum

        Else
          Close filenum
          'In this case the job has finished
        End If


    Case Else
        'This occurs when nothing else does

End Select

End Sub

Sub mnuabout_Click ()
About.Show 1
End Sub

Sub mnuexit_Click ()

'This Quits this application
End

End Sub

Sub mnuiffile_Click ()

MsgBox "This is the Information file Control program.  1 This causes this program to pop the
file name from the top of the current queue,  then test it's status bit and load the next IF
file into the VMD specified.  3 This causes the program to get the filename from the VMD.   4
 This loads this into then increment the status bit then put this record to the next queue.",
 MB_ICONEXCLAMATION

End Sub

Sub mnuqueue_Click ()
MsgBox "This is the Information File control block.  1 Pops the values from the next queue to
 the appropriate text Box's.   2 This Requests information from Fixdmacs to the Next Queue.
 3 Pushes the value from the appropriate text box into the next queue to the end of the queue
.   4 Pushes the value from the appropriate text box into the next queue at the start of the
queue.   6 Pops the values from the current queue to the appropriate text Box's.   7 This req
uests information from Fixdmacs for the current queue.   8 Pushes the value from the appropri
ate text box into the current queue to the end of the queue.  9 Pushes the value from the app
ropriate text box into the current queue at the start of the queue.", MB_ICONEXCLAMATION
End Sub

Sub NoinCqueuetext_Change ()

On Error Resume Next

'When this value changes update the database block in fixdmacs that is resopnsible for queue
control
NoinCqueuetext.LinkPoke

End Sub

Sub NoinNqueuetext_Change ()
On Error Resume Next
```

```
'When this value changes update the database block in fixdmacs that is resopnsible for queue
control
NoinNqueuetext.LinkPoke

End Sub

Sub Picture1_Click ()
 About_Application.Show 1
End Sub

Sub QueControlText_Change ()
' This is the Information File control block
'
' 1 Pops the values from the next queue to the appropriate text Box's
' 2 This Requests information from Fixdmacs to the Next Queue
' 3 Pushes the value from the appropriate text box into the next queue to the end of the queu
e
' 4 Pushes the value from the appropriate text box into the next queue at the start of the qu
eue
' 6 Pops the values from the current queue to the appropriate text Box's
' 7 This requests information from Fixdmacs for the current queue
' 8 Pushes the value from the appropriate text box into the current queue to the end of the q
ueue
' 9 Pushes the value from the appropriate text box into the current queue at the start of the
 queue

On Error Resume Next            'This stops the DDE transfer from tripping the error and stop
ping the program


Select Case quecontroltext.Text


    Case "A" To "Z", "", "a" To "z", "!" To "/", ":" To "@", "[" To "`", "{" To "~", " " To "
                "
        'Any other char that may be input by accident

    Case Is = 1

        'Rather than poping into the current cell VMD I'll try to pop straight into the facto
ry controllers Bidding objects

        'Pop from Next queue

        If Nstartofqueue = NEndofqueue Then
            'Queue is empty
            Exit Sub

        End If

        Nstartofqueue = Nstartofqueue + 1

        If Nstartofqueue > Finishofqueue Then
                'Start of queue is at end of queue wrap around
                Nstartofqueue = Beginofqueue
                End If



        'Set up Manual Links for all of the text box's
        ' Next Queue
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqnopart.A_CV
", NNoofpartstext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqsetup.A_CV"
, Nsetuptimetext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqmachty.A_CV
", NMachinetypetext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqcyctm.A_CV"
, NCycletimetext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqtoolrq.A_CV
", NToolsRequiredtext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqjob.A_CV",
Njobnametext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqiffile.A_CV
", NIFFilenametext)
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nqloadun.A_CV
", NLoadunloadtimetext)
```

NEXTQUE.FRM - 7

```
        'Call CommDDELink(Databasetext.Text & "." & "V" & VMDNumbertext.Text & "nxtqueno.A_CV
", NoinNqueuetext)


        Njobnametext.Text = Nextqueue(Nstartofqueue).job_Name
        NIFFilenametext.Text = Nextqueue(Nstartofqueue).IF_Filename
        NNoofpartstext.Text = Nextqueue(Nstartofqueue).No_ofparts
        Nsetuptimetext.Text = Nextqueue(Nstartofqueue).setup_time
        NMachinetypetext.Text = Nextqueue(Nstartofqueue).machine_type
        NCycletimetext.Text = Nextqueue(Nstartofqueue).Cycle_Time
        NToolsRequiredtext.Text = Nextqueue(Nstartofqueue).tools_required
        NLoadunloadtimetext.Text = Nextqueue(Nstartofqueue).loadunload_time

        ' This Pokes the contents of the text box's to the Data base block specified
        Njobnametext.LinkPoke
        NIFFilenametext.LinkPoke
        NNoofpartstext.LinkPoke
        Nsetuptimetext.LinkPoke
        NMachinetypetext.LinkPoke
        NCycletimetext.LinkPoke
        NToolsRequiredtext.LinkPoke
        NLoadunloadtimetext.LinkPoke
        NoinNqueuetext.LinkPoke

'***Special use****
'This section makes the assumption that when a job is POPed off the next queue into the facto
ry controller
'that a bid request is required to be sent with it

'Send bid request to factory controller
NBidrequesttext.Text = 1
NBidrequesttext.LinkPoke


    Case Is = 2

        'Requesting Information from Fixdmacs
        Njobnametext.LinkRequest
        NIFFilenametext.LinkRequest
        NNoofpartstext.LinkRequest
        Nsetuptimetext.LinkRequest
        NMachinetypetext.LinkRequest
        NCycletimetext.LinkRequest
        NToolsRequiredtext.LinkRequest
        NLoadunloadtimetext.LinkRequest

    Case Is = 3
        'Push to end of Next queue

        If NEndofqueue + 1 = Nstartofqueue Then
            'Queue full
            GoTo fullmessage
            Exit Sub

        End If

        NEndofqueue = NEndofqueue + 1

        If NEndofqueue > Finishofqueue Then
            'If endofqueue at the end of the queue then wrap around
            NEndofqueue = Beginofqueue
         End If

        Nextqueue(NEndofqueue).job_Name = Njobnametext.Text
        Nextqueue(NEndofqueue).IF_Filename = NIFFilenametext.Text
        Nextqueue(NEndofqueue).No_ofparts = NNoofpartstext.Text
        Nextqueue(NEndofqueue).setup_time = Nsetuptimetext.Text
        Nextqueue(NEndofqueue).machine_type = NMachinetypetext.Text
        Nextqueue(NEndofqueue).Cycle_Time = NCycletimetext.Text
        Nextqueue(NEndofqueue).tools_required = NToolsRequiredtext.Text
        Nextqueue(NEndofqueue).loadunload_time = NLoadunloadtimetext.Text

        'Poke number of jobs now in next queue
        NoinNqueuetext.LinkPoke

    Case Is = 4
        'Push to start of Next queue

        If NEndofqueue = Nstartofqueue - 1 Then
```

```
            ' Queue full
            GoTo fullmessage
            Exit Sub

        ElseIf Nstartofqueue < Beginofqueue Then
            Nstartofqueue = Finishofqueue

        End If

        Nextqueue(Nstartofqueue).job_Name = Njobnametext.Text
        Nextqueue(Nstartofqueue).IF_Filename = NIFFilenametext.Text
        Nextqueue(Nstartofqueue).No_ofparts = NNoofpartstext.Text
        Nextqueue(Nstartofqueue).setup_time = Nsetuptimetext.Text
        Nextqueue(Nstartofqueue).machine_type = NMachinetypetext.Text
        Nextqueue(Nstartofqueue).Cycle_Time = NCycletimetext.Text
        Nextqueue(Nstartofqueue).tools_required = NToolsRequiredtext.Text
        Nextqueue(Nstartofqueue).loadunload_time = NLoadunloadtimetext.Text
        Nstartofqueue = Nstartofqueue - 1

        NoinNqueuetext.LinkPoke

    Case Is = 6
        'Pop from current queue

        If Cstartofqueue = CEndofqueue Then
            'Queue is empty
            Exit Sub

        End If

        Cstartofqueue = Cstartofqueue + 1

        If Cstartofqueue > Finishofqueue Then
            'Start of queue is at end of queue wrap around
             Cstartofqueue = Beginofqueue
             End If

        CJobnameText.Text = Currentqueue(Cstartofqueue).job_Name
        CIFFileNameText.Text = Currentqueue(Cstartofqueue).IF_Filename

        ' This Pokes the contents of the text box's to the Data base block specified
        CJobnameText.LinkPoke
        CIFFileNameText.LinkPoke

    Case Is = 7

        CJobnameText.LinkRequest
        CIFFileNameText.LinkRequest

    Case Is = 8
        'Push to end of current queue

        If CEndofqueue + 1 = Cstartofqueue Then
            'Queue full
            GoTo fullmessage
            Exit Sub

        End If

        CEndofqueue = CEndofqueue + 1

        If CEndofqueue > Finishofqueue Then
            'If endofqueue at the end of the queue then wrap around
            CEndofqueue = Beginofqueue
            End If

        Currentqueue(CEndofqueue).job_Name = CJobnameText.Text
        Currentqueue(CEndofqueue).IF_Filename = CIFFileNameText.Text

    Case Is = 9
        'Push to start of queue

        If CEndofqueue = Cstartofqueue - 1 Then
            ' Queue full
            GoTo fullmessage
            Exit Sub

        ElseIf Cstartofqueue < Beginofqueue Then
            Cstartofqueue = Finishofqueue
```

```
        End If

        Currentqueue(Cstartofqueue).job_Name = CJobnameText.Text
        Currentqueue(Cstartofqueue).IF_Filename = CIFFileNameText.Text
        Cstartofqueue = Cstartofqueue - 1

    Case Else
        'This occurs when nothing else does

End Select


' Calculating the number in each queue

If NEndofqueue = Nstartofqueue Then
   NoinNqueuetext.Text = 0

    ElseIf NEndofqueue > Nstartofqueue Then
        NoinNqueuetext.Text = NEndofqueue - Nstartofqueue

    Else
        NoinNqueuetext.Text = (100 - Nstartofqueue) + NEndofqueue

End If

If CEndofqueue = Cstartofqueue Then
   NoinCqueuetext.Text = 0

ElseIf CEndofqueue > Cstartofqueue Then
    NoinCqueuetext.Text = CEndofqueue - Cstartofqueue

Else
    NoinCqueuetext.Text = (100 - Cstartofqueue) + CEndofqueue



End If
Exit Sub

fullmessage:
    Dim msg As String
    msg = VMDNumbertext.Text & " Next queue full"
    MsgBox msg, MB_ICONSTOP, "Queue Status"
    Exit Sub
End Sub



Sub VMDNumberText_Change ()

' This text block contains the name of the VMD number this queue will communicate with

End Sub
```

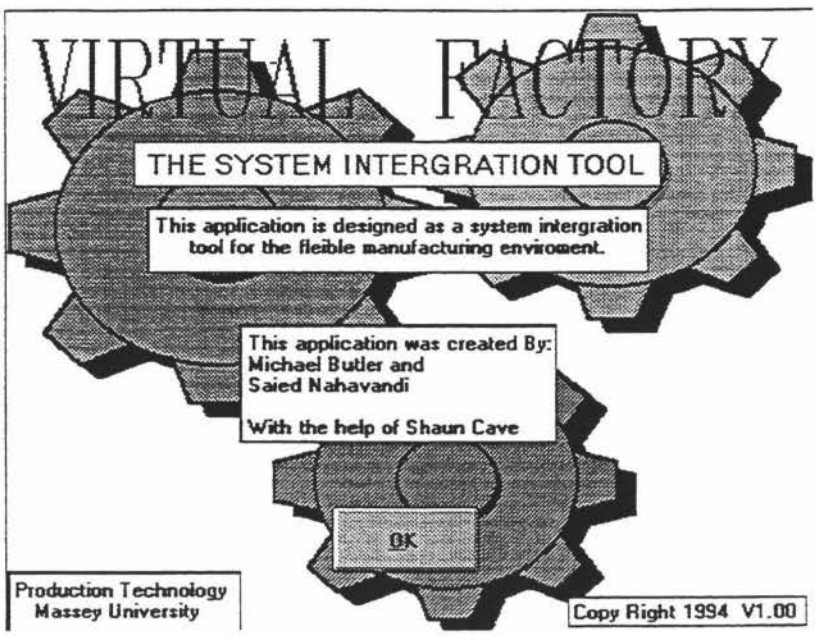
QUEUE1.BAS - 1

```
Option Explicit

'**For queue program

'I don't know why this works but it does, variant's should not work in a type declaration

Type queue

    Job_Name As Variant
    IF_Filename As Variant
    No_ofparts As Variant
    Setup_Time As Variant
    Tools_Required As Variant
    Machine_Type As Variant
    Cycle_Time As Variant
    Loadunload_time As Integer

End Type

' MsgBox parameters
Global Const MB_OK = 0                    ' OK button only
Global Const MB_OKCANCEL = 1              ' OK and Cancel buttons
Global Const MB_ABORTRETRYIGNORE = 2      ' Abort, Retry, and Ignore buttons
Global Const MB_YESNOCANCEL = 3           ' Yes, No, and Cancel buttons
Global Const MB_YESNO = 4                 ' Yes and No buttons
Global Const MB_RETRYCANCEL = 5           ' Retry and Cancel buttons

Global Const MB_ICONSTOP = 16             ' Critical message
Global Const MB_ICONQUESTION = 32         ' Warning query
Global Const MB_ICONEXCLAMATION = 48      ' Warning message
Global Const MB_ICONINFORMATION = 64      ' Information message

Global Const MB_APPLMODAL = 0             ' Application Modal Message Box
Global Const MB_DEFBUTTON1 = 0            ' First button is default
Global Const MB_DEFBUTTON2 = 256          ' Second button is default
Global Const MB_DEFBUTTON3 = 512          ' Third button is default
Global Const MB_SYSTEMMODAL = 4096         'System Modal

' MsgBox return values
Global Const IDOK = 1                     ' OK button pressed
Global Const IDCANCEL = 2                 ' Cancel button pressed
Global Const IDABORT = 3                  ' Abort button pressed
Global Const IDRETRY = 4                  ' Retry button pressed
Global Const IDIGNORE = 5                 ' Ignore button pressed
Global Const IDYES = 6                    ' Yes button pressed
Global Const IDNO = 7                     ' No button pressed


'** For file IF control
'
Type jobrecord

    jobname As String * 13
    currentindex As Integer
    Finaltask As Integer
    tasknumber As Integer
    Machinetype As Integer
    taskname As String * 13
    Estcycletime As Integer
    Setuptime As Integer
    Estloadunloadtime As Integer
    Noofcycles As Integer
    Machinefilename As String * 13
    Geometricfilename As String * 13
    Toolsrequired As Integer
    Fixturesfilename As String * 13
    Status As Integer
    duedate As String * 8
    force As Integer

End Type

' Define job record type and global file management variables
Global Recordname As jobrecord  'Record structure
Global temprecord As jobrecord

' Status variables for file manipulation
Global filestatus As Integer
Global position As Integer
```

QUEUE1.BAS - 2

```
Global filenum As Integer


Function loadApplication (Appname, Path) As Variant

'Written by Michael Butler
'Department of Production Technology
'Massey University
'
'This function takes an application name and path then attempts to start it
'if any errors occur it aborts, returns loadapplication yes or no ie it has been loaded
'
'
'Improvement can be made here by testing if the application is open before it is opened using
 an API call
'

Dim response, x                                         'response from message box
Const IDYES = 6, IDNO = 7                               'return from message box

    On Error GoTo ErrorHandel   'if an error occurs goto errorhandel
    x = Shell(Path & Appname)
    loadApplication = True

Exit Function

ErrorHandel:

    response = MsgBox("Is " & Appname & " currently running?", MB_YESNOCANCEL, "Application S
tatus")      'Ask if want to start application
    If response = IDYES Then
        loadApplication = True
    Else
        MsgBox "Error in opening application, this operation may not continue", 48, "Applicat
ion Status"
        loadApplication = False
    End If
    Exit Function

End Function

Sub readfromfile (record As jobrecord)
'read from file at position into recordname
If position = 0 Then
    position = 1
    End If

Get filenum, position, record

End Sub

Sub savetofile (record As jobrecord)

Dim response As Integer

' save all information to file
     Put #filenum, position, record

 End Sub
```

# Information about the Next Queue application



ABOUT.FRM - 1

```
Sub Command1_Click ()
    Unload about
End Sub

Sub Command2_Click ()
Unload about

End Sub
```

# The Help window for the Next Queue application

Created By Michael Butler
Department of Production Technology, Massey University

This application is a QUEUEing program designed to transfer data via DDE Links to and from this appliation and FIXDMACS. This enables data to be Pushed onto this Queue and Poped off again using the Queue control Box.

These functions are used throught the Queue Control text Box

1 - This will Pop data from the Next Queue
2 - This requests data from Fixdmacs, must be used before a Push
3 - This will Push data on to the end of the Next Queue
4 - This Pushes data on the the start of the Next Queue

6 - This Pops data from the Current Queue
7 - This requests data from Fixdmacs, must be used before Push
8 - This Pushes data to the end of the Current Queue
9 - This Pushes data to the start of the Current Queue

OK

ABOUTAPP.FRM - 1

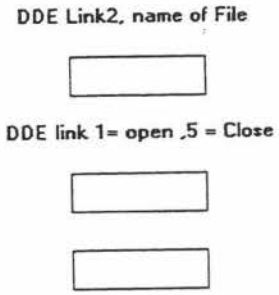
```
Option Explicit

Sub Command1_Click ()
About_Application.Hide
End Sub
```

# Appendix G

## Program Listing for File Com Visual Basic Program

This appendix contains the program listing and respective print out of the windows that the File Com application consists of.

## Main Window for the File Com application




The main window consists of three text box's, only the top two are active. The top box is used to enter the name of the file to be down loaded via a communications port. The second text box is used to control the transmission of the file.

```
FILE_COM.FRM - 1

Option Explicit

Dim filename                                    'for communication port 2, this is the file n
ame
Dim CommparametersPort2, CommHandshakePort2     'For communication poarameters
Dim LinkItem                                    'Item for DDE to link with
Const Close_DDE = 5, Open_DDE = 1               'Constants for 1 and 5
Dim checkfilename                               'For checking the file is still the same

Sub CommDDELink (LinkItem1, LinkItem2, LinkItem3)

'This procedure is designed to run once on startup to set up a DDE link with fixdmacs
'this consists of two DDE links, which block names are passed in as Linkitem1 & 2
'link item one here is the control block, with link item 2 being the filename block
'
Dim firsttime As Double                         'This is the variable for the timer function
Const NONE = 0, Link_Manual = 2, Link_Automatic = 1
firsttime = Timer

    Do Until Timer > firsttime + 2
        'This is a timing loop which will loop for four secounds, to wait for the DDE server
to open compleatly
    Loop


If DDETextBox1.LinkMode = NONE Then             'Open link 1
        On Error Resume Next
        DDETextBox1.LinkTopic = "DMDDE|DATA"
        DDETextBox1.LinkItem = LinkItem1
        DDETextBox1.LinkMode = Link_Automatic
        End If

firsttime = Timer
    Do Until Timer > firsttime + 2
        'This is a timing loop which will loop for four secounds, to wait for the DDE server
to open compleatly
    Loop

If DDETextBox2.LinkMode = NONE Then             'If not already open
        On Error Resume Next
        DDETextBox2.LinkTopic = "DMDDE|DATA"    'set topic
        DDETextBox2.LinkItem = LinkItem2        'set link item, this is passed in
        DDETextBox2.LinkMode = Link_Automatic   'link mode is automatic
        End If

If DDETextBox3.LinkMode = NONE Then              'open link3
        On Error Resume Next
        DDETextBox3.LinkTopic = "DMDDE|DATA"
        DDETextBox3.LinkItem = LinkItem3
        DDETextBox3.LinkMode = Link_Automatic
        End If


On Error GoTo Errorhandel
If CInt(DDETextBox1.Text) = Close_DDE Then          'Close DDE server, 5 = close DDE link
DDETextBox1.LinkMode = NONE                         'Close the link
DDETextBox2.LinkMode = NONE

AppActivate "FIX DMACS DDE Server"
SendKeys "%", True
SendKeys "{ENTER}", True
SendKeys "{UP}", True
SendKeys "{UP}", True
SendKeys "{ENTER}", True
MsgBox " File transfer Compleate DDE Server Closed", 64, "Status"

End If

Exit Sub

Errorhandel:
    Exit Sub
End Sub

Sub CommOutPort2 (filename, CommparametersPort2, CommHandshakePort2)

'This routinewhen called, reads a file then outputs it via the comport at the given communica
tion
'paramenters. The file name, communication parameters and handshaking mode are passed in.
```

FILE_COM.FRM - 2

```
'If their are more than 6000 characters then this process should wait until it is possible to
'transmitte again.
'This has not been written yet

Dim stringcount As Integer, TransmitChar, Blar            'Declare variables

' The communication parameters are set in the main program so as not to cause an run time err
or
Comm2.PortOpen = True                                                    'Enable comport
Comm2.Settings = CommparametersPort2                                     'Set at 9600 Baud, no par
ity, 8 dat 1 stop bit
Comm2.Handshaking = CommHandshakePort2                                   'Xon/Xof

If Len(filename) Then 'If file contains something
    If filename <> checkfilename Then
    filename = "b:\" & filename & ".txt"
    Else
    End If
    checkfilename = filename
    Open filename For Input As #1                                        'Open file
    Do While Not EOF(1)
            TransmitChar = Input(1, #1)                                  'Get char from file one c
har at a time
            Comm2.Output = TransmitChar                                  'Put out char via comport
 one char at a time
            stringcount = stringcount + 1

            If stringcount = 6000 Or CInt(DDETextBox3.Text) = 1 Then      'Transmitte only 30
00 char at a time
                If stringcount = 6000 Then
                    Blar = MsgBox("Transmittion halted, waiting", 48, "Communication Status")
                Else
                    Do Until CInt(DDETextBox3.Text) = 2                   'Stop and wait un
til ready.
                        Blar = MsgBox("Transmittion halted, waiting", 48, "Communication Stat
us")
                        stringcount = 0
                    Loop
                End If
                stringcount = 0
            End If
    Loop
        Blar = MsgBox("Transmision Compleate", 48, "Communication Status")
        Close #1                                                         'Close file
End If
Comm2.PortOpen = False                                                   'Close comport1

End Sub

Sub Comm2_OnComm ()
'This is the communications driver control
End Sub

Sub DDETextBox1_Change ()

'This text box contans control information from fixdmacs
'this box initiates a communications job by waiting for a change event
'when the value of this box changes and it is a 1 the communication is started.
'
On Error GoTo exitsub                          'if error occurs exit subroutine

If CInt(DDETextBox1.Text) = Open_DDE Then      'if the value of this text bos is 1 then do

    CommparametersPort2 = "9600,n,8,1"
    CommHandshakePort2 = 2                      'Hardware handshake
    Call CommOutPort2(filename, CommparametersPort2, CommHandshakePort2)    'This is a call t
o the communications procedure.
End If

If CInt(DDETextBox1.Text) = Close_DDE Then      'if a 5 is recieved then close DDE
    Unload CommDDefile_form                     'and unload this form closing this applicatio
n
End If
Exit Sub

exitsub:                                        'in case of an error
    Exit Sub

End Sub
```

FILE_COM.FRM - 3

```
Sub DDETextBox2_Change ()
'
'This is the secound text block which contains a link to a text block which contains
'the name of the file to be loaded form the b:\drive

If DDETextBox2.Text <> "" Then
    filename = DDETextBox2.Text    'lets the value of this box = the filename variable
End If

End Sub

Sub DDETextBox3_Change ()
' This is the third dde link if this value is 1 then the flow of char will be halted if
' it is 2 then it will begin again
End Sub

Sub Form_Load ()
'
'
'This application is written By Michael Butler
'This application looks at two data base blocks if fixdmacs
'VBControl1 and VB control2 for the control and file name information
'VBControl1 = 1 start DDE and transfer file at 9600,n,8,1  Xon/Xoff
'VBControl1 = 5 closes DDE link and server, and other value has no effect.
'VBControl3 = 1 stop char transmittion, if = 2 start transmition again

'DDETextBox1.Visible = False                                    'This hides the DDE text box
's used by the DDE link for recieving control info
'DDETextBox2.Visible = False
'DDETextBox3.Visible = False
'CommDDefile_form.Visible = False                               'Hides Main form window
load_file.Visible = False

' Set Comport 2 Communication parameters

Comm2.CommPort = 2                                               'Set comport2


If loadApplication("DMDDE.exe", "c:\wdmacs\") = True Then  'Loads dde server

    Call CommDDELink("IEL1.VBControl1.F_CV", "IEL1.VBControl2.A_CV", "IEL1.VBControl3.F_CV")
    'Sets up DDE link
AppActivate "Com2 Driver"
End If
End Sub

Sub Load_File_Click ()

'This is just a test button which will prompt the user to enter a file name and
'will then load that file
'These set the communication parameters of comport2

filename = InputBox("Please Enter File Name in B drive")
CommparametersPort2 = "9600,n,8,1"
CommHandshakePort2 = 1

Call CommOutPort2(filename, CommparametersPort2, CommHandshakePort2)

End Sub
```

FILE_COM.BAS - 1

```
Option Explicit

Dim Appname As String, Path As String 'Application name and path



Function loadApplication (Appname, Path) As Variant

'Written by Michael Butler
'Department of Production Technology
'Massey University
'
'This function takes an application name and path then attempts to start it
'if any errors occur it aborts, returns loadapplication yes or no ie it has been loaded
'
'
'Improvement can be made here by testing if the application is open before it is opened using
 an API call
'

Dim response, x                                         'response from message box
Const IDYES = 6, IDNO = 7                               'return from message box

    On Error GoTo ErrorHandel   'if an error occurs goto errorhandel
    x = Shell(Path & Appname)
    loadApplication = True

Exit Function

ErrorHandel:

    response = MsgBox("Is " & Appname & " currently running?", 4, "Application Status")
'Ask if want to start application
    If response = IDYES Then
        loadApplication = True
    Else
        MsgBox "Error in opening application, this operation may not continue", 48, "Applicat
ion Status"
        loadApplication = False
    End If
    Exit Function

End Function
```

# Appendix H

## Fisher & Paykel PSC Program Code and Diagrams

This appendix contains the program used within the Fisher and Paykel PSC. These programs also have been explained diagrammatically with the use of functional diagrams.

## PSC Programming

### Main program

The main program is where some of the states and flags are declared. It was attempted to keep all of the declarations that could be kept local to be done so, thus only flags that will be required globally have been declared here.

This is also where the other loops are started, some of the loops require to be started straight away, others are triggered when a flag is set true. This will probably mean that the main program will simply be a controller of the other loops.

### OutputPortcontrol

This loop monitors the Request?Outport_? flag for each bit of the port, when this flag is set true then the corresponding output port is switched on as well as the Current?Outport? flag, if the request flag is false then the port is switched off. and the current flag is reset.

If the Resetalloutports flag is set then all outputs are switched off and all of the current flags are reset.

### ComportControl1

This module is presently for receiving data via comport 1. This module will eventually have code which will enable it to both transmit and receive data. When theInputCommunicationRequest flag is set, then the main state starts this loop. This loop then initialises then communication port and sets the communication parameters, which are presently fixed but could easily be altered. The CommunicationInput1 state then tests if the last char received was a EOF char, if not then characters are read from the comport one at a time and placed into the memory starting at address 120.

### ComportControl3

This module is for output of chars from the memory of the PSC via comport 3. It does this in a similar way to the input function. By reading characters one at a time and outputting them via comport 3 and doing this until the EOF or ^@ is reached. At present their are several limitations to these two modules in their current form. The communications parameters are set and cant be altered easily by the user, this will require a link to Fix to provide this information. Secondly the length of the file to be stored in the PSC is only limited, with only 60 Kbytes available. Finally both of these communications routines require a loop to monitor and control the communication transfer, ie splitting up the files into several pieces for transfer.

# Functional Diagram of Main PSC program

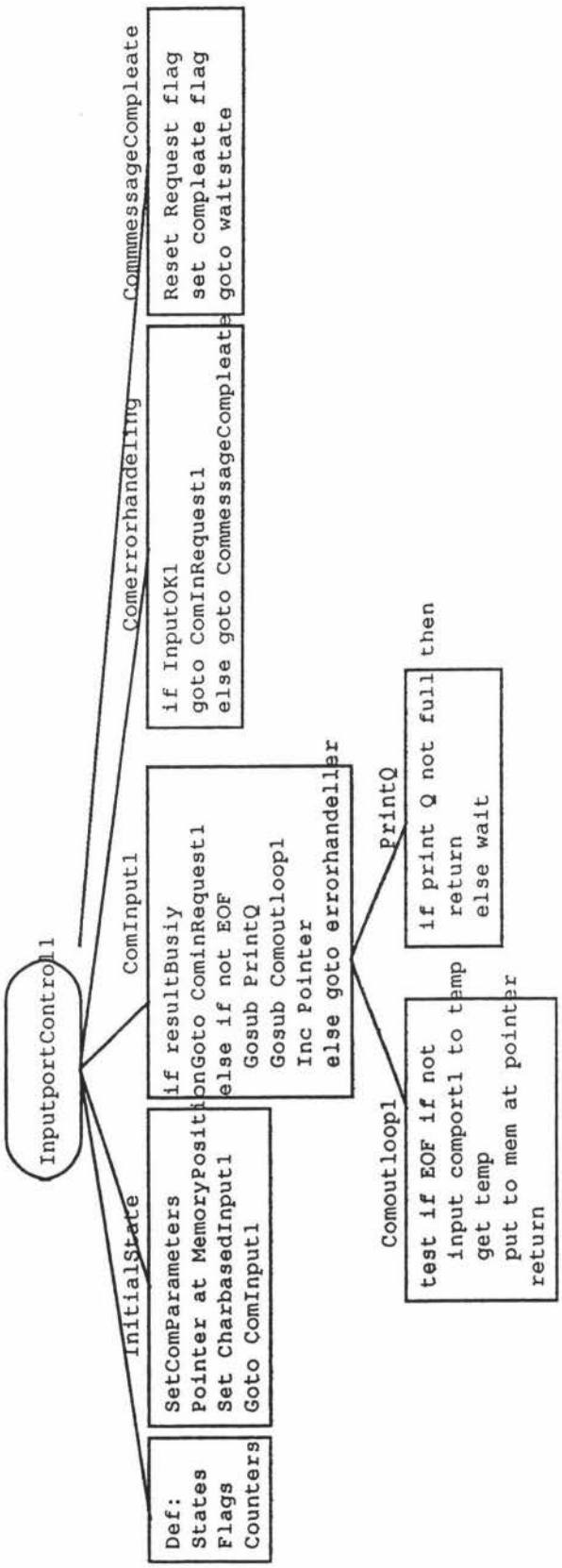
# Main PSC Program

```
;
; Michael Butler
; 22/10/93
;
;
; The Main program reserves states 71 to 99
;
def state 51 as MainchoiceOut
;def state 52 as
def state 73 as CommunicationWaitstate
def state 71 as SetVariablesOut
def state 61 as MainchoiceIn
def state 100 as Initialstate3
def state 104 as CommunicationOutputRequest3
def state 111 as CommunicationInputRequest1
def state 112 as Initialstate1
;
; **Definitions for communications**
;
def comport 3 as CommunicationPort3
def comport 2 as CommunicationPort2
def comport 1 as CommunicationPort1
;
;
; **Definitions for flags**
;
; *Flags for Comport Communication
;
def flag 1571 as ResultBusiy                ;Set high if result register is to be used.
def flag 1572 as ForceResultBusiy           ;Set high if Result register if forced cleared
def flag 1573 as CommunicationNotPossible   ;Set high if communication out one of the comports is
not possible
def flag 1574 as CommunicationRequestOut3   ;Set high if communication is requested
def flag 1575 as CommunicationPort3Busiy    ;Set high if port is in use
;
def flag 1936 as InputCommunicationRequest1  ;Set high if input is requested via comport1
def flag 1937 as InputCommunicationCompleate1 ;Set high if input has been correctly recieved via
comport1
def flag 1938 as CommunicationPort1Busiy    ;Set high if comport1 busiy
def flag 1939 as  MessageInMemory1          ;Set if a mesage is in memoryspace 1
def flag 1940 as ActiveMemory1              ;Set if message is to go into memory space 1
;
; *Flags for input ports
;
;  Current state of input ports
def flag 1553 as Current1Inport_0           ;Set high if a signal is recieved on input port 1.0
def flag 1554 as Current1Inport_1
def flag 1555 as Current1Inport_2
def flag 1556 as Current1Inport_3
def flag 1557 as Current1Inport_4
def flag 1558 as Current1Inport_5
def flag 1559 as Current1Inport_6
def flag 1560 as Current1Inport_7
;
; *Flags for output ports
;
; General Flags
def flag 830 as Resetalloutports            ;Set high if all output ports are reset
```

```
def flag 831 as Erroralloutports          ;Set high if general error on output ports
;
; Output Port 3.0 to 3.6
;                                  Request for output flags
def flag 801 as Request3Outport_0
def flag 802 as Request3Outport_1
def flag 803 as Request3Outport_2
def flag 804 as Request3Outport_3
def flag 805 as Request3Outport_4
def flag 806 as Request3Outport_5
def flag 807 as Request3Outport_6
def flag 808 as Request3Outport_7
;
;                                  Current state of output port
def flag 810 as Current3Outport_0
def flag 811 as Current3Outport_1
def flag 812 as Current3Outport_2
def flag 813 as Current3Outport_3
def flag 814 as Current3Outport_4
def flag 815 as Current3Outport_5
def flag 816 as Current3Outport_6
def flag 817 as Current3Outport_7
;                                  Interlocking flag, if high then ouput OK
def flag 820 as Interlock3OPutport_0
def flag 821 as Interlock3OPutport_1
def flag 822 as Interlock3OPutport_2
def flag 823 as Interlock3OPutport_3
def flag 824 as Interlock3OPutport_4
def flag 825 as Interlock3OPutport_5
def flag 826 as Interlock3OPutport_6
def flag 827 as Interlock3OPutport_7
;
;
; **Definitions for loops**
;
def loop 1 as Mainloop
def loop 2 as OutputPortcontrol
def loop 3 as InputPortcontrol
def loop 4 as Comportcontrol3
def loop 5 as Comportcontrol1
;
; **Starting of loops**
;
START Mainloop AT SetVariablesOut
START OutputPortcontrol AT MainchoiceOut
START InputPortcontrol AT MainchoiceIn
;
; **Definitions for SD files**
;
def SD 001 as outputportprocedures
def SD 002 as inputportprocedures
def SD 003 as comportprocedures3
def SD 004 as comportprocedures1

SetVariablesOut                    ;Test state
     SET Request3Outport_0
     SET Request3Outport_3
     SET InputCommunicationRequest1
```

```
    ;SET CommunicationRequestOut3
    goto CommunicationWaitstate

;
CommunicationWaitstate
; Communication port controlling procedure if CommunicationRequestOutput3 or
InputCommunicationRequest
; are set to high then the releted loop will be started.
;

    RESET CommunicationNotPossible
    RESET ResultBusiy
    SET ActiveMemory1              ;Set to use memory space1

if InputCommunicationRequest1          ;If their is an input communication request for com 1
  AND ActiveMemory1                    ;and In memory area 1 then start communication loop in 1
  SET MessageInMemory1
  ONCE
  goto CommunicationInputRequest1
  END

if CommunicationRequestOut3            ;If memory area 1 active and a output communication
request
  AND ActiveMemory1                    ;for com 3 then start out com 3.
  RESET MessageInMemory1
  ONCE
  goto CommunicationOutputRequest3
  END

;
CommunicationOutputRequest3
            if CommunicationRequestOut3
            RESET CommunicationRequestOut3
            START Comportcontrol3 AT Initialstate3
            goto CommunicationWaitstate
            else
              SET CommunicationNotPossible
              PRINT CommunicationPort3 "Communication Error3"
              goto CommunicationWaitstate


CommunicationInputRequest1
            RESET InputCommunicationRequest1
            START Comportcontrol1 AT Initialstate1
            goto CommunicationWaitstate
```

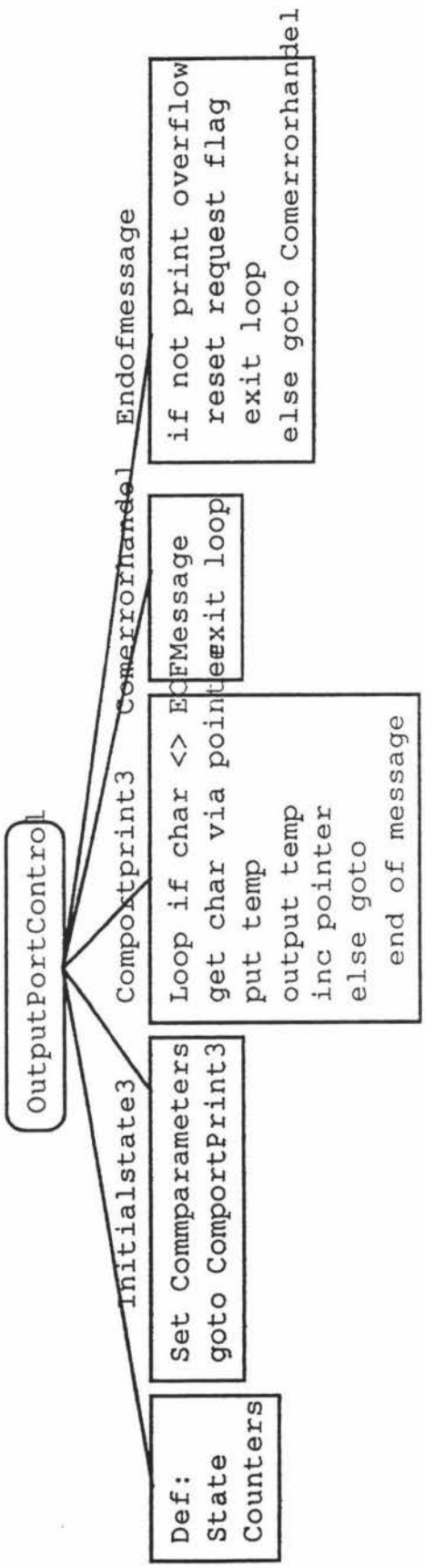
# Functional diagram for the input communication port communication program



# Input communications control program

```
;
;
;
; Micheal Butler
; Created on 29/10/93
; Modified on 1/11/93
;
; This module is designed for recieving RS-232 communication from a computer or terminal
; This program uses Comport 1.
;
; This program recives data from comport1 when InputCommunicationRequest flag is set to high
; it then reads in char and places them one char at a time in to memory
; starting at address MEM 120, when a ^@ is encountered the InputCommunicationCompleate flag
; is set and the loop is exited.
;
;
; This module reserves flags 1936 to 2016
;                counters 127 to 137
;                states 111 to 120
;
;
;
def state 113 as PrintQue1
def state 114 as CommunicationInput1
def state 115 as CommMessageCompleate1
def state 117 as CommunicationErrorHandelling1
def state 118 as CommunicationOutputLoop1
;
def mem 103 as Tempararyswapaddress1 BYTE
;
def counter 127 as MemoryPointer1
def counter 128 as LastValueTest1
;
; Other Flags for use
; InputCommunicationRequest
; InputCommunicationCompleate
; CommunicationPort1Busiy
;
Initialstate1
        COMPORT 1 96 0 8 011H              ;Setting and flushing comport1
        FLUSH CommunicationPort1
        SET EchoComport1
        POINT MemoryPointer1 AT MemoryPosition   ;Point Pointer at start of memory location
        SET RESULT = 999                   ;Set so wont trigger exit from loop first time
        SET LastValueTest1 = 999
        SET CharBasedInput1                ;Set for charicter based input on comport1
        goto CommunicationInput1

;
CommunicationInput1
            if ResultBusiy                 ;ResultBusiy Flag goes high if RESULT register is in use
              OR CommunicationPort1Busiy
              PRINT CommunicationPort1 "Communication error, Comport 1 Busiy"
              goto CommunicationInputRequest1
              else
                if RESULT <> 60            ;Test for ^@ at end of file
                AND LastValueTest1 <> 94
                SET ResultBusiy
```

## Functional diagram for the output communication port communication program



# Output communications control program

```
;
;
; Michael Butler
; Created 25/10/93
; Modified 29/10/93
;
; This module is a Com Port communication procedure for PSC CNC communication.
;
; This program reserves state 100 to 120
;                flags 1570 to 1664
;                counters 100 to 120
;
; Definitions for the ouput routine
;
def state 101 as ComPortprint3
def state 102 as EndofMessage3
def state 103 as CommunicationErrorHandeling3
;
def const 0 as AsciiNul 0                    ;This is the Ascii null Char
def counter 101 as MemoryPointer3            ;Points to current position in memory
;
def MEM 100 as TempararySwapaddress3 BYTE    ;Temparary is the variable use to output the
char to the com port
def MEM 120 as MemoryPosition STRING         ;This is the start of the string, Next address wiil
be 180
;
; Preset Error flags for Comports InputOK0,
; InputNotOK0, PrintOFComport0.
;
;                              This is the initialisation state
Initialstate3


      COMPORT 3 96 0 8 011H          ;Points memorypointer at start of string
      POINT MemoryPointer3 AT MemoryPosition
      SET RESULT = 999               ;Stops null ascii test from tripping
        goto ComPortprint3


;                            This state outputs the
;                            message starting at MemoryPosition and ending at AsciiNul or ^@
ComPortprint3
      if ResultBusiy                 ;If results register is busiy
        AND NOT CommunicationPort3Busiy
         PRINTLN CommunicationPort3 "RESULT Register Error, Error may occur"
         SET ForceResultBusiy        ;Forced to take control of results register
         RESET ResultBusiy

      else
        IF RESULT <> AsciiNul        ;Tests if char currently read is end of message,
                                     ;Possible problems here should sence ^@
         SET ResultBusiy
         PAUSE LOOP 2
         PAUSE LOOP 3
         GET BYTE VIA MemoryPointer3 ;Reads char from memory and outputs it
com comport3
         PUT BYTE TempararySwapaddress3
```

```
                GOSUB PrintQue1
                GOSUB CommunicationOutputLoop1
                INC MemoryPointer1          ;Increment Pointer to next position
                RESET ResultBusiy
                else
                  goto CommunicationErrorHandelling1
;
CommunicationErrorHandelling1                    ;This routine handels the case of the last charicter
incorrectly recieved
                  if NOT InputOK1
                  PRINT CommunicationPort1 "Error has Occured in Conversion"
                  goto CommunicationInputRequest1
                  else
                  goto CommMessageCompleate1
;
CommMessageCompleate1                            ;Message correctly recieved
              RESET InputCommunicationRequest1
              SET InputCommunicationCompleate1
              goto CommunicationWaitstate        ;Exit Back to calling state
;
PrintQue1
    if Comport1TXCount < 470                     ;Check that input buffer is not to full
    RETURN
;
CommunicationOutputLoop1                         ;Communication loop which inputs charicters into
memory address
    XCHG                                 ;Puts results register into LastValueTest1 for ^@ test in
earler state
    GET BYTE Tempararyswapaddress1
    PUT LastValueTest1
    XCHG
    INPUT CommunicationPort1 Tempararyswapaddress1 "" ;Inputs single char into
Temararyswapaddress
              GET BYTE Tempararyswapaddress1  ;Places this from Temaray.. to Memory
pointed at by Memory pointer
              PUT BYTE VIA MemoryPointer1
              if InputOK1                    ;Returns if recieved correctly
               RETURN
              if InputNotOK1
                goto CommunicationErrorHandelling1
```

```
            PRINTLN CommunicationPort3 TempararySwapaddress3
            INC MemoryPointer3
            RESET ResultBusiy
            goto CommunicationErrorHandeling3
           else
             goto EndofMessage3
;
;                                        This state tells you that the message is compleate
EndofMessage3
      if NOT PrintOFComport3                          ;Tests preset flag
      PRINTLN CommunicationPort3 "Message complete"
      RESET CommunicationRequestOut3
      goto CommunicationWaitstate                     ;Returns control calling state
      else
        goto CommunicationErrorHandeling3

;
;                                        This state handels a comport overflow flag
CommunicationErrorHandeling3
      if PrintOFComport3
      PRINTLN CommunicationPort3 "Printer Overflow error^@"
      goto CommunicationWaitstate                     ;Exits on error back to calling state
                                       ;Must include notification of error to PSC and user
      else
        goto ComPortprint3•
```

```
else
  RESET Current1Inport_7
```

# Input control program

```
;
; Michael Butler
; Created on 24/10/93
; Modified on 25/10/93
;
; This SD file is for the control of the input functions of the PSC
; This state cuurently will operate for the input module 1 only.
; The 020 module will take ports 2,3 and 4, the 050 module will use ports
; 5 to 11 two input ports for each 12 bit channel.
;
; Reserve flags 1568 to 1570
; Reserve counters 81 to 97
; Reserve states 61 to 65
;
; definitions for input ports
;
def IP 1.0 as In1Port_0
def IP 1.1 as In1Port_1
def IP 1.2 as In1Port_2
def IP 1.3 as In1Port_3
def IP 1.4 as In1Port_4
def IP 1.5 as In1Port_5
def IP 1.6 as In1Port_6
def IP 1.7 as In1Port_7
;
;
MainchoiceIn
        if In1Port_0
          SET Current1Inport_0
          else
            RESET Current1Inport_0
        if In1Port_1
          SET Current1Inport_1
          else
            RESET Current1Inport_1
        if In1Port_2
          SET Current1Inport_2
          else
            RESET Current1Inport_2
        if In1Port_3
          SET Current1Inport_3
          else
            RESET Current1Inport_3
        if In1Port_4
          SET Current1Inport_4
          else
            RESET Current1Inport_4
        if In1Port_5
          SET Current1Inport_5
          else
            RESET Current1Inport_5
        if In1Port_6
          SET Current1Inport_6
          else
            RESET Current1Inport_6
        if In1Port_7
          SET Current1Inport_7
```

# Output control program

```
;
; Created on 22/10/93
; Michael Butler
;
; Modified on 24/10/93
;
; This SD file is a Library for output functions
; These functions will be used similarly to those in C
; By calling the master entity in this loop which will
; direct the current variables to the correct Output
; state.
;
; This SD file reserves Flags 800 to 976
; This SD file reserves counters 60 to 80
; This SD file reserves states 50 to 60
;
; Define flags two for each output port
; The first flag will be it's current state as defined by the state in this
; loop, their will also be another flag for the state the port actually holds.
; Their will be one flag to account for the interlocks, these flags have been
; defined in the main program
;
;
; state 51 is defined as MainchoiceOut
;
def state 52 as Errorcheck
def state 54 as Interlockoutput
def state 53 as Outportshutdown
;
; Output Port 3 port definitions
;
def op 3.0 as Out3Port_0
def op 3.1 as Out3Port_1
def op 3.2 as Out3Port_2
def op 3.3 as Out3Port_3
def op 3.4 as Out3Port_4
def op 3.5 as Out3Port_5
def op 3.6 as Out3Port_6
def op 3.7 as Out3Port_7
;
;
;
MainchoiceOut
if Request3Outport_0
  Out3Port_0 on
  SET Current3Outport_0
else
  Out3Port_0 off
  RESET Current3Outport_0
end

if Request3Outport_1
  Out3Port_1 on
  SET Current3Outport_1
else
  Out3Port_1 off
```

```
  RESET Current3Outport_1
  end

if Request3Outport_2
  Out3Port_2 on
  SET Current3Outport_2
else
  Out3Port_2 off
  RESET Current3Outport_2
  end

if Request3Outport_3
  Out3Port_3 on
  SET Current3Outport_3
else
  Out3Port_3 off
  RESET Current3Outport_3
  end

if Request3Outport_4
  Out3Port_4 on
  SET Current3Outport_4
else
  Out3Port_4 off
  RESET Current3Outport_4
  end

if Request3Outport_5
  Out3Port_5 on
  SET Current3Outport_5
else
  Out3Port_5 off
  RESET Current3Outport_5
  end

if Request3Outport_6
  Out3Port_6 on
  SET Current3Outport_6
else
  Out3Port_6 off
  RESET Current3Outport_6
  end

if Request3Outport_7
  Out3Port_7 on
  SET Current3Outport_7
else
  Out3Port_7 off
  RESET Current3Outport_7
  end

goto Errorcheck
;
; In the case of any errors this state checks for a reset all ports command
; or an emergincy shut down command.
;
Errorcheck
      if Resetalloutports
```

```
        or Erroralloutports
        ops all off
        RESET Current3Outport_0
        RESET Current3Outport_1
        RESET Current3Outport_2
        RESET Current3Outport_3
        RESET Current3Outport_4
        RESET Current3Outport_5
        RESET Current3Outport_6
        RESET Current3Outport_7
end
        if Erroralloutports
          goto Outportshutdown
        else
        goto MainchoiceOut
;
; This is an emergincy shut down state, any special procedures must be
; undertaken here.
;
Outportshutdown

 PAUSE OutputPortcontrol


.
```

# Appendix I

## Review of Manufacturing and SCADA Software

**Summary of process control and monitoring software features (adapted from Control Engineering 1990)**

| Company | Product | Date | Price | Process control | Data Acquisition | SQC | SPC | Batch control | On line changes | Network Capability |
|---|---|---|---|---|---|---|---|---|---|---|
| GE Fanuc | Cimplicity system 3000 model 1 | 1989 | 17000-45000 | X | X | | | | X | X |
| | Cimplicity system 3000 model | 1989 | 14800 | X | X | | | | X | X |
| Gray Soft | WHIPS | | | X | X | X | | | X | |
| Hewlett-Packard | HP 75000 PC Data Acquisition system | 1989 | 10400 | X | X | | | X | X | |
| IBM | Plantworks | 1989 | | X | X | X | X | X | X | X |
| Industrial Control Specialists | Hot line | 1986 | 3000 | X | X | | | X | X | X |
| Lawson Labs Inc | LOG 512 | | X | | | | | | | |
| **Intellution** | **The FIX** | **1984** | **4000** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| | **FIX DMACS** | **1988** | **5500** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| Micro Speciality Systems | RTM 3500 | 1986 | 5000 | X | X | X | X | X | X | X |
| Opto 22 | Cyrano | 1989 | 1195 | X | X | | | X | X | X |
| Sprecher & Schuh Automation | Supervisory Micro Control | 1985 | 50000 | X | X | X | X | X | X | X |
| Taylor Industrial Software | Process Window | 1986 | 2500 | X | X | | | X | X | |
| Zontec Inc | LogLink | 1989 | | X | X | X | X | X | | X |
| U.S. Data Corp. | Factory Link | 1986 | 2500 | X | X | | X | | X | X |

Manufacturing Software Database Search: Mon, Nov 7, 1994 3:22:11 PM
Department Of Production Technology
Massey University
Private Bag
Palmerston North
Ph (06) 356-9099 Fax (06) 350-5604

---

**Criteria Used For Search:**

••Application Areas Include:

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
  11 Bill Of Materials Processor
  12 Operations Details (STD Route Sheets)
  14 Tooling Specifications
20 ORDER CONTROL
  22 Purchasing
30 STOCK CONTROL
  31 Re-ordering
  32 Forecasting
40 PRODUCTION PLANNING
  41 MRP-Materials Requirements Planning
  42 Capacity Planning
  44 JIT-Just In Time
  45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
  51 Job Location and Status (Job Tracking)
60 COSTING/PRODUCTION ACCOUNTING
  61 Job Costing
  63 Budgeting
70 PLANT CONTROL
  71 Numerical Control Systems(NC, CNC)
  72 Process Control (including PLC control)
  73 Planned Maintenance
100 CAD/CAM
  101 Graphics
  102 Draughting
  103 Special Purpose (eg PCB design)
  105 CAM Aids (eg Part Program Generation)

---

**Results of the Search Contained 12 Records**

**********

• Package Name: Chameleon/Chameleon 2000
• Record ID Number: 17
• Match with Specified Criteria: 80 %
• Contact Name: G.A. Nightingale
• Company Name and Address: GenasysLtd, PO Box 7287, Wellesley St, Auckland.
• Phone: (09)3095430 • Fax: (09)3732679
• Typical Package Cost: *
• Notes on Cost of Package: Subject to number of users
eg 8 Users 22 modules =$94,700

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
70 PLANT CONTROL
73 Planned Maintenance
90 OTHER MANUFACTURING OPTIONS
93 EDI-Electronic Data Interchange
100 CAD/CAM
104 CAD/CAM Links

• Operating Systems: .
• Hardware and Software Requirements and Costs: All types of Unix Workstations
eg 20 users = 32Mb RAM 1Gb Disk, 486 or 88100 Chip
Requirements and hence costs are subject to users needs
• Support and Maintenance Services: 20% of list price per annum
• Current Number of Users: NZ: 8 Total: 2500

---

• Package Name: TIMS
• Record ID Number: 31
• Match with Specified Criteria: 80 %
• Contact Name: Stephen Ryder
• Company Name and Address: Geac Computers NZ Ltd, P.O.Box 2733, Auckland.
• Phone: (09)3091860 • Fax: (09)3079243
• Typical Package Cost:
• Notes on Cost of Package: TIMS is priced on the number of Concurrent Users Licensed, A full system would start at around $40,000 installed

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
70 PLANT CONTROL
72 Process Control (including PLC control)
73 Planned Maintenance
74 Data Capture
80 QUALITY SYSTEMS
81 Total Quality Management
90 OTHER MANUFACTURING OPTIONS
92 Project Planning/Control (PERT/CPM)
93 EDI-Electronic Data Interchange
100 CAD/CAM
104 CAD/CAM Links

• Other Areas of Application: Includes interactive sales and resource planning.
Fully addresses job, batch, discrete and process environments.
SPC users third party software
PLC control requires customised interface- Can be integrated to give a CIM environment
• Operating Systems: UNIX all versions, AIX.
• Hardware and Software Requirements and Costs: IBM 486's for up to 15 workstations. Also Open Systems Compliant.
From 15-200 Terminals requires large Unix machines.
Others include: IBM RS6000 HP, ALTOS, NCR, APPLE, COMPAQ
System Size dependent on user reqts
Requires Unix, Aix, or SCO Xenix.
• Support and Maintenance Services: Annual Support Fee of 15% of initial licence fee provides full support services including:Help desk support, Modem support, New Releases, Bug fixes, Documentation and general guidance and assistance to optimise TIMS software.
• Further Information and Comments: Full Consultation, Training and review services provided.
Help desk at main centres.
System configurable for manufacturing, distribution and financials.
User interface individually customisable.
Designed, written and supported in NZ.
Most widely used software of its type in NZ.
Head office in Wellington. Sold throughout NZ, Australia and Asia.

Other Offices:
Noel Noonan
Private Bag 2696
Christchurch
Ph(03)3663942

Fax(03)3657762

Ken Templer
PO Box 57044
Mana
Wellington
Ph(04)2338186
Fax(04)2339285
• Current Number of Users: NZ: 170 Total: 320
---

• Package Name: MANMAN/X
• Record ID Number: 23
• Match with Specified Criteria: 70 %
• Contact Name:
• Company Name and Address: UXL Computer Systems Ltd, PO Box 6368, Wellesley St, Auckland.
• Phone: (09)3098585 • Fax: (09)3078279
• Typical Package Cost: 70,000 *
• Notes on Cost of Package: $70,000 to $120,000 for typical installation.

•• Areas of Application ••
1 PRODUCTION CONTROL PACKAGE 2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL 11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets) 13 Engineering Changes Control
14 Tooling Specifications 20 ORDER CONTROL
21 Sales Order Entry and Control 22 Purchasing
30 STOCK CONTROL 31 Re-ordering
32 Forecasting 40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning 42 Capacity Planning
43 Scheduling 44 JIT-Just In Time
50 WORK IN PROGRESS CONTROL 51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing 53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING 61 Job Costing
62 Standard Costing 63 Budgeting
64 Quoting/Estimating 90 OTHER MANUFACTURING OPTIONS
92 Project Planning/Control (PERT/CPM) 93 EDI-Electronic Data Interchange
100 CAD/CAM 104 CAD/CAM Links
• Other Areas of Application: Other Modules offer Service,Project Manufacture, Lot Control
• Operating Systems: SCO Unix, HP-UX, AIX, OSF, MPE-iX, Open VMS.
• Hardware and Software Requirements and Costs: IBM RS6000, HP9000, HP3000, DEC VAX, DEC ALPHA.
Any Hardware running SCO Unix
Cost vary between $20,000 to $250,000 depending on the number of users
Requires Database:eg ISAM, INGRES, ORACLE.
• Support and Maintenance Services: Service and support contract offered
• Further Information and Comments: MANMAN/X is from ASK Computer Systems, an ASK GROUP Company.
Full Policies and Procedures Package available, Integrated Software and Customisable
---

• Package Name: PRO:MAN
• Record ID Number: 22
• Match with Specified Criteria: 73 %
• Contact Name: Graham Fallwell
• Company Name and Address: Software Application Ltd, PO Box 9847, Newmarket, Auckland.
• Phone: (09)3094003 • Fax: (09)5756292
• Typical Package Cost: 25000
• Notes on Cost of Package: 3 Users: $25000
6 Users: $30000

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
80 QUALITY SYSTEMS
81 Total Quality Management
82 Statistical Process Control
100 CAD/CAM
104 CAD/CAM Links

• Operating Systems: Advanced PICK on, Dos, Unix, OS/2, Windows NT, Universe, PI/OPEN.
• Hardware and Software Requirements and Costs: IBM 386, 486, 4Mb RAM 210Mb HD
Typical Costs $7k-$30k
8 or 16 port serial card eg DIGIBOARD
• Support and Maintenance Services: $125/Month
$3000 New Releases
• Further Information and Comments: 4GL Screens and Enquires
Multi-Warehouses
Distribution Requirements Planning
Focus Forecasting
Currently in use at A and G Price Ltd, machine shop Thames. (See NZ Engineering News article Oct 1992)
• Current Number of Users: NZ: 18 Total: 1500+

---

• Package Name: IBM(NZ) Ltd-Various Packages
• Record ID Number: 907
• Match with Specified Criteria: 100 %
• Contact Name: Richard Malloch
• Company Name and Address: Manufacturing Account Manager, IBM (NZ) Ltd, PO Box 1219, Christchurch.
• Phone: (03) 3792840 • Fax: (03)3728928
• Typical Package Cost:

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
70 PLANT CONTROL
71 Numerical Control Systems(NC, CNC)
72 Process Control (including PLC control)
73 Planned Maintenance
74 Data Capture
75 Robotics Controls
80 QUALITY SYSTEMS
81 Total Quality Management
82 Statistical Process Control
90 OTHER MANUFACTURING OPTIONS
91 Operations Research (LP, Networks etc)
92 Project Planning/Control (PERT/CPM)
93 EDI-Electronic Data Interchange
100 CAD/CAM
101 Graphics
102 Draughting
103 Special Purpose (eg PCB design)
104 CAD/CAM Links
105 CAM Aids (eg Part Program Generation)

• Other Areas of Application: IBM has access to worldwide applications to suit any customers needs.
The majority of software can be tailored specifically for your environment.
• Operating Systems: AS/400, RS/6000, AIX.
• Hardware and Software Requirements and Costs: IBM RS/6000
IBM AS/400
• Further Information and Comments: IBM and its business partners have thousands of applications that run on either the RS/6000 or AS/400 computers.
Please enquire for more details.
---

• Package Name: MAPICS/XA
• Record ID Number: 908
• Match with Specified Criteria: 80 %
• Contact Name: Olwyn McConnell
• Company Name and Address: Madison Systems Ltd, PO Box 8279, Symonds Street, Auckland.
• Phone: (09)3093655 • Fax: (09)3790685
• Typical Package Cost: 50000+

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
64 Quoting/Estimating
70 PLANT CONTROL
71 Numerical Control Systems(NC, CNC)
72 Process Control (including PLC control)
73 Planned Maintenance
74 Data Capture
90 OTHER MANUFACTURING OPTIONS
93 EDI-Electronic Data Interchange
100 CAD/CAM
104 CAD/CAM Links

• Other Areas of Application: Has full Distribution (ie Order entry-Shipping-Invoicing) and full financials INTEGRATED with manufacturing applications.
Also Warehouse management, Production analysis, Multiple site control
• Operating Systems: .
• Hardware and Software Requirements and Costs: IBM AS/400
Typically $70,000+
Some modules can be run on PC under a LAN environment with an AS/400.
• Support and Maintenance Services: Hardware maintenance- $300 +per month depending on size.
• Further Information and Comments: Interfaces with IBM Office Vision-(Word Processing, Calenders etc)
Product Info available on request.
• Current Number of Users: NZ: 10 Total: 13000

---

• Package Name: PRISM
• Record ID Number: 909
• Match with Specified Criteria: 70 %
• Contact Name: Chris Johnson
• Company Name and Address: Prism Software Ltd, PO Box 8684, Symonds St, Auckland.
• Phone: (09) 3584441 • Fax: (09) 3582144
• Typical Package Cost:
• Notes on Cost of Package: Between $5,000 and $100,000
Dependent on modules installed and size of machine.

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
70 PLANT CONTROL
74 Data Capture
80 QUALITY SYSTEMS
81 Total Quality Management
82 Statistical Process Control
90 OTHER MANUFACTURING OPTIONS
93 EDI-Electronic Data Interchange

• Other Areas of Application: Fully Integrated Accounting software available as well
• Operating Systems: Dos, Windows, OS/2, Novell, UNIX, VMS.
• Hardware and Software Requirements and Costs: IBM PC
Min 386SX 1Mb RAM 40 MB HD
Approx $5000
• Support and Maintenance Services: $1000 Support and upgrade contract
• Further Information and Comments: Geared for Printing, Packaging and Plastics Industries
• Current Number of Users: NZ: 205 Total: 210

---

• Package Name: CONTROL:Manufacturing
• Record ID Number: 47
• Match with Specified Criteria: 83 %
• Contact Name: John Titchener
• Company Name and Address: CINCOM Systems of NZ Ltd, PO Box 3842, Wellington 1.
• Phone: (04)4723655 • Fax: (04)4712712
• Typical Package Cost:
• Notes on Cost of Package: Dependant upon hardware, O/S, Hardware Capacity, and No. of Modules.
IBM mainframe running MVS $500,000+
Digital VAX running VMS $300,000 to $450,000+
UNIX $95,000 to $300,000.

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
70 PLANT CONTROL
71 Numerical Control Systems(NC, CNC)
72 Process Control (including PLC control)
74 Data Capture
90 OTHER MANUFACTURING OPTIONS
92 Project Planning/Control (PERT/CPM)
93 EDI-Electronic Data Interchange
100 CAD/CAM
104 CAD/CAM Links

• Other Areas of Application: Customisable-Source code supplied .
Through CONTROL:Manufacturing's open interface a number of third party products provide input to CONTROL: in the form of graphics and CAD/CAM.
Supports discrete, repetitive, process and project-oriented manufacturing techniques. Is especially suited to mixed-mode manufacturing and highly regulated markets such as pharmaceutical and aerospace/defence.
• Operating Systems: MVS, VSE, VMS, UNIX.
• Hardware and Software Requirements and Costs: IBM Mainframe, DEC VAX or Alpha, UNIX hardware vendors, eg HP, DEC etc.
Typically $70,000 to $90,000 for UNIX and small VAX.
Mainframe and large VAX open ended.
Licences for CINCOM's 4GL (MANTIS) and RDBMS (SUPRA) are embedded.
• Support and Maintenance Services: Annual usage charge set at 15-20% of initial fees
• Further Information and Comments: CONTROL: Manufacturing is written in Cincon's fourth generation language(MANTIS) and based on Cincom's SUPRA RDBMS. Source code is provided at no additional charge.
CONTROL: is especially suited to mixed mode manufacturing and highly regulated markets such as pharmaceutical, chemical, and aerospace/defence.
• Current Number of Users: NZ: 1 Total: 600+

---

• Package Name: CA-PRMS
• Record ID Number: 49
• Match with Specified Criteria: 76 %
• Contact Name: Mark Moss
• Company Name and Address: Customer Support Manager, Computer Associates (NZ) Ltd, PO Box 997, Wellington.
• Phone: (04) 8017654 • Fax: (04)8017655
• Typical Package Cost:
• Notes on Cost of Package: Based on machine size and no of users.

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
62 Standard Costing
70 PLANT CONTROL
71 Numerical Control Systems(NC, CNC)
72 Process Control (including PLC control)
73 Planned Maintenance
74 Data Capture
75 Robotics Controls
80 QUALITY SYSTEMS
81 Total Quality Management
82 Statistical Process Control
90 OTHER MANUFACTURING OPTIONS (PERT/CPM)
92 Project Planning/Control
93 EDI-Electronic Data Interchange

• Other Areas of Application: Plant Control options available on Spec.
CA-PRMS has a CIM interface facility which transfers data from process control to the main business application. "ONSPEC" and "POMS" interfaced to PRMS.
• Operating Systems: .
• Hardware and Software Requirements and Costs: IBM AS/400
Maintracker is the suggested plant maintenance scheduling application.
• Support and Maintenance Services: Fully supported by Computer Associates
• Further Information and Comments: CA-PRMS is an advanced manufacturing system designed specifically for the IBM AS/400 system
CA-PRMS completely integrates production planning, plant operations, customer service, distribution and financial management.
CA-PRMS offers a firm and flexible foundation for CIM synergy.
CA-PRMS accommodates emerging co-existent environments by providing any combination of discrete, repetitive and process manufacturing. The product also supports multinational and international needs.

---

• Package Name: Dynamic Distribution/Manufacturing
• Record ID Number: 910
• Match with Specified Criteria: 76 %
• Contact Name: Micheal McKenzie
• Company Name and Address: System Dynamics Corporation (NZ) Ltd, PO Box 2205, Auckland.
• Phone: (09)3584842 • Fax: (09)3585325
• Typical Package Cost: 50000
• Notes on Cost of Package: Typically in the range of $50,000 to $500,000.

•• Areas of Application ••

2 CUSTOMIZABLE CONTROL
10 PRODUCTION INFORMATION
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
70 PLANT CONTROL
73 Planned Maintenance
90 OTHER MANUFACTURING OPTIONS
92 Project Planning/Control (PERT/CPM)
93 EDI-Electronic Data Interchange
100 CAD/CAM
104 CAD/CAM Links

• Operating Systems: DOS, LAN's (under Novell), UNIX.
• Hardware and Software Requirements and Costs: Any UNIX or DOS based hardware. Requires Oracle RDBMS and Tools
• Support and Maintenance Services: Full support available
• Current Number of Users: NZ: 2 Total: 100+

---

• Package Name: BPCS
• Record ID Number: 60
• Match with Specified Criteria: 70 %
• Contact Name: G Cooksley
• Company Name and Address: SSA Pacific (NZ) Ltd, PO Box 6622, Wellesley St, Auckland.
• Phone: (09)3580555 • Fax: (09)3580444
• Typical Package Cost:
• Notes on Cost of Package: POA

•• Areas of Application ••

1 PRODUCTION CONTROL PACKAGE
2 CUSTOMIZABLE
10 PRODUCTION INFORMATION CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
45 Constraint/Bottleneck Management
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
64 Quoting/Estimating
80 QUALITY SYSTEMS
81 Total Quality Management
90 OTHER MANUFACTURING OPTIONS
93 EDI-Electronic Data Interchange

• Other Areas of Application: Fully Integrated manufacturing System including: Financials, distribution, logistics, export documentation, IMASE support.
• Operating Systems: .
• Hardware and Software Requirements and Costs: IBM AS/400
Typically $100,000+
• Support and Maintenance Services: Helpline, ongoing support/maintenance and training.
• Further Information and Comments: New Product releases /enhancements are supplied at 6 month intervals.
System fully integrated across all modules.
• Current Number of Users: NZ: 200 Total: 7000

---

• Package Name: JD Edwards
• Record ID Number: 57
• Match with Specified Criteria: 70 %
• Contact Name: David Copplestone
• Company Name and Address: JD Edwards NZ Ltd, PO Box 33-130, Takapuna, Auckland.
• Phone: (09)4860866 • Fax: (09)4860679
• Typical Package Cost: 250000
• Notes on Cost of Package: Starts at $250,000 depending on the number of modules and processor model.
•• Areas of Application ••

2 CUSTOMIZABLE CONTROL
11 Bill Of Materials Processor
12 Operations Details (STD Route Sheets)
13 Engineering Changes Control
14 Tooling Specifications
20 ORDER CONTROL
21 Sales Order Entry and Control
22 Purchasing
30 STOCK CONTROL
31 Re-ordering
32 Forecasting
40 PRODUCTION PLANNING
41 MRP-Materials Requirements Planning
42 Capacity Planning
43 Scheduling
44 JIT-Just In Time
50 WORK IN PROGRESS CONTROL
51 Job Location and Status (Job Tracking)
52 Factory Documentation Printing
53 Shop Floor Data Collection (eg Bar-codes)
60 COSTING/PRODUCTION ACCOUNTING
61 Job Costing
62 Standard Costing
63 Budgeting
70 PLANT CONTROL
73 Planned Maintenance
74 Data Capture
90 OTHER MANUFACTURING OPTIONS
93 EDI-Electronic Data Interchange

• Other Areas of Application: Specific software written for the major oil companies to meet their own requirements is available in addition to our standard manufacturing modules.
• Operating Systems: .
• Hardware and Software Requirements and Costs: IBM AS/400
Memory 24Mb+ dept on no of users.
HD 3000MB+ dept on volume of data.
Wide choice of workstations, and networked PC's.
EG AS/400 model F10 from $100,000+
Financial software modules also available.
• Support and Maintenance Services: JDE maintenance 12% of package software list price, new releases, fixes.
• Further Information and Comments: Of the users: 2 NZ and 200 total use the manufacturing modules.
JDE is a fully integrated manufacturing, distribution and financial suite of application software for the AS/400.
JDE's software is a good match in the discrete manufacturing area, for companies turnover of $100+ million, or subsidiaries of multinational manufacturing organisations.
JDE NZ is an IBM remarketter and is able to supply complete installation of software, IBM hardware and support services.
Written using a 4GL called WORLDCASE which generates RPGIII code for the AS/400.
• Current Number of Users: NZ: 26 Total: 2500

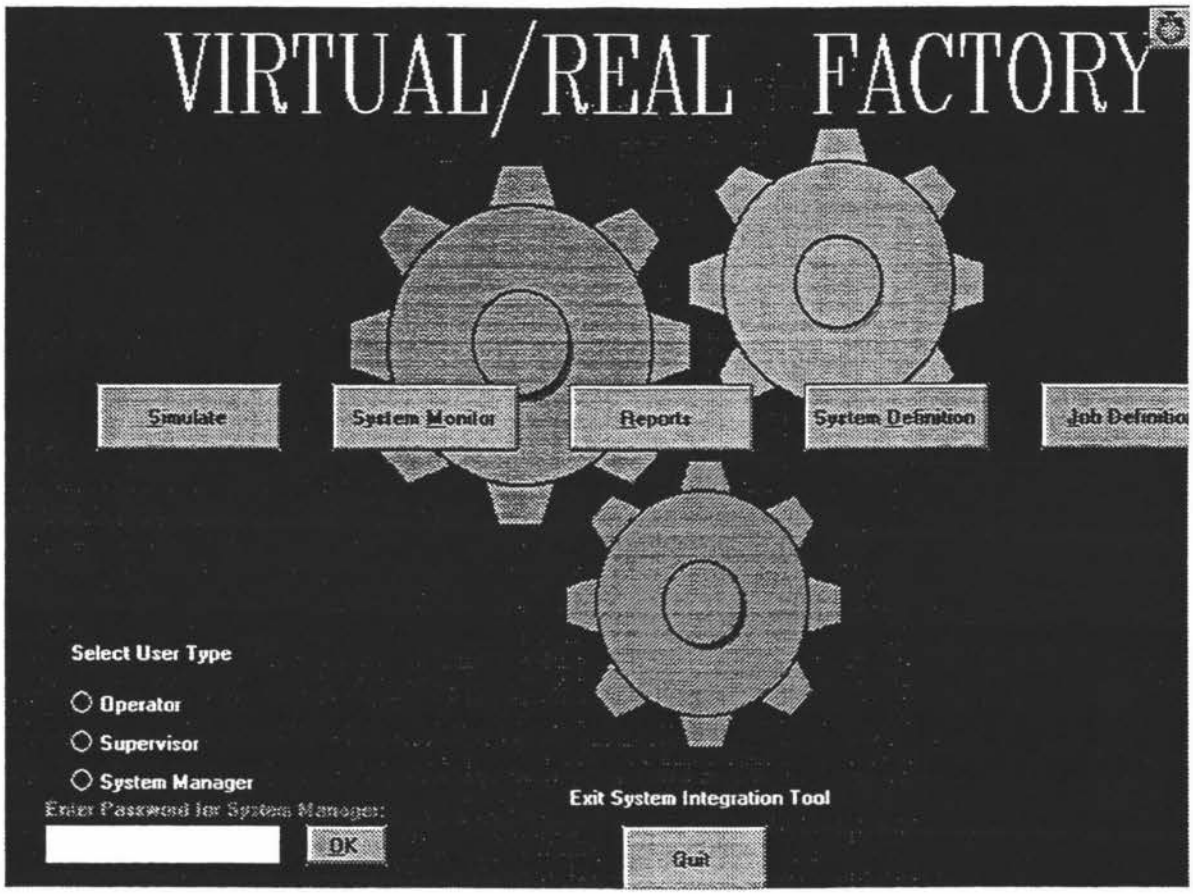
---

# Appendix J

## Program Listing for Virtual Factory Visual Basic Program

This appendix contains the program listing and respective window for the Virtual Factory application. The Virtual Factory application is the major Visual Basic application with which the user interacts with in order to utilise the virtual factory.

## Virtual Factory main menu window



FACT_MAN.FRM - 1

```
Sub deletefromdatefile (record As daterecord, position As Integer)

'Pass in the record structure and the position from which to delete from

' This deletes a record from the file
    record.date = ""
    record.filename = ""
    Put datefilenum, position, record

End Sub

Sub writetoscreen (record As jobrecord)
' This procedure writes the data in the fields of jobrecord type passsed into this procedure
' to the input fields of the next Queue_form

' To let user Know when a job has been precessed
Beep

queue_form!NJobNameText.text = record.jobname
queue_form!NCycleTimeText.text = record.estcycletime
queue_form!NSetuptimeText.text = record.setuptime
queue_form!NMachineTypeText.text = record.Machinetype
queue_form!NNoofPartsText.text = record.noofcycles
queue_form!NIFFileNameText.text = record.jobname
queue_form!NToolsRequiredText.text = record.toolsrequired
queue_form!NLoadunloadtimetext.text = record.Estloadunloadtime


End Sub

Sub Command1_Click ()
Const MB_ICONEXCLAMATION = 48
    If (StrComp(Text1, "Soup_4", 0) = 0) Then
        Command1.Visible = False
        Text1.Visible = False
        Label1.Visible = False
        'MsgBox "Correct Password Entered", 64, Title
        Text1.text = ""
        user = "Supervisor"
        ElseIf (StrComp(Text2, "Soup_4", 0) = 1 Or -1) Then
        Beep
        MsgBox "Invalid password. ", 48
        Text1.text = ""
    End If
End Sub

Sub Command2_Click ()
    If (StrComp(Text2, "VMD_4", 0) = 0) Then
        Command2.Visible = False
        Text2.Visible = False
        Label2.Visible = False
        'MsgBox "Correct Password Entered", 64, Title
        Text2.text = ""
        user = "Manager"

    ElseIf (StrComp(Text2, "VMD_4", 0) = 1 Or -1) Then
        Beep
        MsgBox "Invalid password. ", 48, Title
        Text2.text = ""
    End If
End Sub

Sub Execute_Tasks_Button_Click ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994
system_Monitor.Show 1
End Sub

Sub Exit_Button_Click ()
    End
End Sub
```

FACT_MAN.FRM - 2

```
Sub Form_Load ()
'
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

Unload start_form

Load queue_form

user = "Operator" 'Default initial value

End Sub

Sub Job_Definition_Button_Click ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

  job_Definition_Screen.Show 1
End Sub

Sub mnuabout_Click ()
    About.Show 1
End Sub

Sub mnuEnd_Fix_Click ()
    AppActivate "FIX DMACS Startup"
    SendKeys "%{F4}", True
    fixrunning = "No"
    mnuStart_Fix.Enabled = True
    mnuEnd_Fix.Enabled = False
End Sub

Sub mnuExecute_Tasks_Click ()

system_Monitor.Show 1

End Sub

Sub mnuexit_Click ()
    End
End Sub

Sub mnuHow_To_Click ()
    How_To.Show 1
End Sub

Sub mnuReports_Schedular_Click ()
If (user = "Manager" Or user = "Supervisor") Then
    Reports_Schedular_Screen.Show 1
Else
    Beep
    MsgBox "Only System Manager or Supervisor may process reports", 16, Title
End If


End Sub

Sub mnuSimulate_Click ()

Dim Response                    'Variable to see if Fix is running
                                'Check if Fix is running...
If fixrunning = "No" Then       'If not, inform user
    Response = MsgBox("FIX DMACS is not currently running. Do you wish to start it now?", 4,
Title)
    If Response = IDYES Then
        X = Shell("C:\WDMACS\FIX.EXE", 1)
        fixrunning = "Yes"
        mnuStart_Fix.Enabled = False
        mnuEnd_Fix.Enabled = True
    Else
        MsgBox "Sorry Fixdmacs is not operating", 32, Title
    End If
```

```
ElseIf fixrunning = "Yes" Then 'If it is, start simulation
    X = Shell("C:\WDMACS\VIEW.EXE", 1)
    SendKeys "%", True
    SendKeys "{ENTER}", True
    SendKeys "{ENTER}", True
    SendKeys "simstart.odf", True
    SendKeys "{ENTER}", True
End If

End Sub

Sub mnuStart_Fix_Click ()
    X = Shell("C:\WDMACS\FIX.EXE", 1)
    fixrunning = "Yes"
    mnuStart_Fix.Enabled = False
    mnuEnd_Fix.Enabled = True
End Sub

Sub mnuSupport_Click ()
    Product_Support.Show 1
End Sub

Sub mnuSystem_Definition_Click ()

Const NONE = 0, Link_Manual = 2

Dim Response                    'Variable to see if Fix is running
If user = "Manager" Then                                    'Check if Fix is running...
    If fixrunning = "No" Then       'If not, inform user
        Response = MsgBox("FIX DMACS is not currently running. Do you wish to start it now?",
 4, Title)
        If Response = IDYES Then
            X = Shell("C:\WDMACS\FIX.EXE", 1)
           fixrunning = "Yes"
            mnuStart_Fix.Enabled = False
            mnuEnd_Fix.Enabled = True
        Else
            MsgBox "Sorry Fixdmacs is not operating", 32, Title
        End If
    ElseIf fixrunning = "Yes" Then
        system_definition.Show 1
    End If
Else
    Beep
    MsgBox "Only System Manager may define the system", 16, Title
End If
End Sub

Sub mnuTask_Definition_Click ()
job_Definition_Screen.Show 1
End Sub

Sub mnuUseHelp_Click ()
    UseHelp.Show 1
End Sub

Sub Reports_Schedular_Button_Click ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994


If (user = "Manager" Or user = "Supervisor") Then
    Reports_Schedular_Screen.Show 1
Else
    Beep
    MsgBox "Only System Manager or Supervisor may process reports", 16, Title
End If

End Sub



Sub Simulate_Button_Click ()
'This application was written By:
'Michael Butler
'Department of Production Technology
```

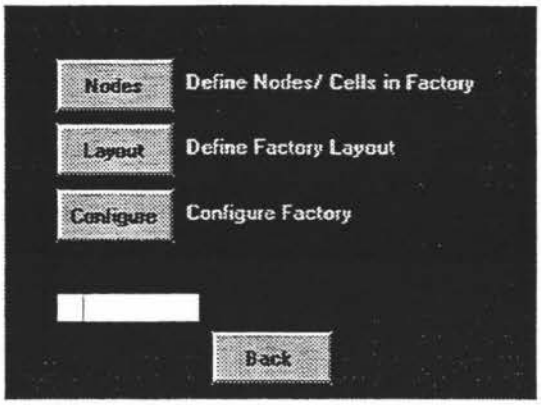
```
'Massey University
'Copyright 1994

If testfix = IDYES Then

Dim Response                    'Variable to see if Fix is running
                                'Check if Fix is running...
If fixrunning = "No" Then       'If not, inform user
    Response = MsgBox("FIX DMACS is not currently running. Do you wish to start it now?", 4,
Title)
    If Response = IDYES Then
        X = Shell("C:\WDMACS\FIX.EXE", 1)
        fixrunning = "Yes"
        mnuStart_Fix.Enabled = False
        mnuEnd_Fix.Enabled = True
    Else
        MsgBox "Please Try Again", 32, Title
    End If
ElseIf fixrunning = "Yes" Then 'If it is, start simulation
    X = Shell("C:\WDMACS\VIEW.EXE", 1)
    SendKeys "%", True
    SendKeys "{ENTER}", True
    SendKeys "{ENTER}", True
    SendKeys "simstart.odf", True
    SendKeys "{ENTER}", True
End If
End If

End Sub

Sub System_Definition_Button_Click ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

system_definition.Show 1

End Sub

Sub Text1_KeyPress (KeyAscii As Integer)
    If KeyAscii = 13 Then 'Check for enter key
        If (StrComp(Text1, "Soup_4", 0) = 0) Then
            Command1.Visible = False
            Text1.Visible = False
            Label1.Visible = False
            'MsgBox "Correct Password Entered", 64, Title
            Text1.text = ""
            user = "Supervisor"

        ElseIf (StrComp(Text2, "Soup_4", 0) = 1 Or -1) Then
            MsgBox "Invalid password. ", 48
            Text1.text = ""
        End If
    End If
End Sub

Sub Text2_KeyPress (KeyAscii As Integer)
    If KeyAscii = 13 Then 'Check for enter key
        If (StrComp(Text2, "VMD_4", 0) = 0) Then
            Command2.Visible = False
            Text2.Visible = False
            Label2.Visible = False
            'MsgBox "Correct Password Entered", 64, Title
            Text2.text = ""
            user = "Manager"

        ElseIf (StrComp(Text2, "VMD_4", 0) = 1 Or -1) Then
            MsgBox "Invalid password. ", 48, Title
            Text2.text = ""
        End If
    End If
End Sub

Sub Timer1_Timer ()


'This procedure scans at regular intervals, interval set to 60 secounds
```

FACT_MAN.FRM - 5

```
'when a timer event occurs, the file default directory datefile.txt' is scaned record by reco
rd for any jobs
'with the date the same or less then todays date, ie is it's due date todays or a date gone p
ast
'if so then this job is loaded and put into the next queue of the factory manager.

On Error Resume Next

'Declartions for datefile program
'dim datefile as daterecord
Dim dateposition As Integer

datefilenum = FreeFile

' Opening file date file program
Open defalutpath & "datefile.txt" For Random Shared As datefilenum Len = Len(datefile)
dateposition = 1


'*Delcarations for IF file transfer between the file specified in the date file and the queue
ing program
Dim tempjobrecord As jobrecord
Dim jobposition As Integer
Dim recordlen As Integer
Dim jobfilenum As Integer

    '*** This is a very basic scheduling method***
    '*** A job is released into the system when it's due date to todays date or it's due date
 is less than todays date, ie it was ment to be made yesterday***

' Search file until until reached end of file
Do Until EOF(datefilenum)
    'Get date from first record in file
    Get datefilenum, dateposition, datefile

    If Trim(datefile.date) = "" Then
        'This is to catch an empty record field in a record
        dateposition = dateposition + 1

    'Compare date in record with todays date
    ElseIf DateDiff("d", datefile.date, Date) >= 0 Then
        'release this job into the system
        'load information from file and place into queue

        ' Set file control variables
        jobrecordlen = Len(tempjobrecord)
        jobfilenum = FreeFile

        'open IF file specified in datefile.filename field.
        Open defaultpath & Trim(datefile.filename) For Random Shared As jobfilenum Len = jobr
ecordlen
        jobposition = 1

        'get first task in file record
        Get jobfilenum, jobposition, tempjobrecord

        'load infromation from file into next queue
        Call writetoscreen(tempjobrecord)
        'Push onto end of the next queue
        queue_form!QueControlText.text = 0
        queue_form!QueControlText.text = 3

        'deletes record from datefile
        Call deletefromdatefile(datefile, dateposition)
        Close jobfilenum

    End If

    dateposition = dateposition + 1
    Get datefilenum, dateposition, datefile

Loop
Close datefilenum
End Sub

Sub User_type_Click (Index As Integer)
If User_type(Index).Index = 1 Then
    Command1.Visible = True
    Text1.Visible = True
```

```
    Label1.Visible = True
    Text1.SetFocus
ElseIf User_type(Index).Index = 2 Then
    Command2.Visible = True
    Text2.Visible = True
    Label2.Visible = True
    Text2.SetFocus
End If
End Sub
```

# System Definition main window



```
FACTORY_.FRM - 1

Option Explicit

Const Close_DDE = 5, Open_DDE = 1, NONE = 0, Link_Manual = 2, Link_Automatic = 1
  'Constants for 1 and 5

'Delcare variable for the list of nodes
Dim nodelistfile As nodelistrecord

'Declare a variable for the node definition file
Dim nodefile As noderecord


Sub CommDDELink (LinkItem, DDETextName As TextBox)

'This procedure is designed to run once on startup to set up a DDE link with fixdmacs
'The Block name is passed in as Linkitem and the DDE Text box name is passed in as
' DDETextName



If DDETextName.LinkMode = NONE Then              'If not already open
        On Error Resume Next
        DDETextName.LinkTopic = "DMDDE|DATA"     'set topic
        DDETextName.LinkItem = LinkItem          'set link item, this is passed in
        DDETextName.LinkMode = Link_Manual    'link mode is automatic
        End If


On Error GoTo Errorhandel

Exit Sub

Errorhandel:
    MsgBox " An Error Has Occured in Setting up the DDE Links! "
    Exit Sub
End Sub

Sub Back_Click ()
Unload Factory_Definition

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub Layout_Click ()
define_Layout.Show 1

End Sub

Sub mnuabout_Click ()
about.Show 1

End Sub

Sub mnuexit_Click ()
Unload Factory_Definition

End Sub

Sub mnuusing_Click ()
usehelp.Show 1

End Sub

Sub Nodes_Click ()
define_Nodes.Show 1

End Sub

Sub Setup_Click ()
```

FACTORY_.FRM - 2

```
'This procedure sets up the factory, ie all the factory definition information is transmitted
 via DDE to the nodes specified

Dim nodelistfilenum As Integer
Dim nodelistposition As Integer

Dim nodefilenum As Integer
Dim nodeposition As Integer


On Error Resume Next

Dim response As Integer

Call testfix

'Is fixdmacs running is not then give message
If fixrunning = IDNo Then
    MsgBox "Unable to Setup Factory, Fixdmacs must be running on this Node", MB_ICONEXCLAMATI
ON, "SETUP FACTORY"

    Else
        Beep
        'Fixdmacs is running, tell user that this is an important operation and must be under
taken carfully
        response = MsgBox("FACTORY SETUP reconfigures the layout of the Nodes specified previ
ously in the Factory Layout Definition, this operation will alter the Virtual Layout of the F
actory, do you wish to continue?", MB_YESNO, "SETUP FACTORY")

        If response = IDYES Then

            'set position of nodelistfile to the first record
            nodelistposition = 1

            nodelistfilenum = FreeFile

            'open list of nodes file
            Open defaultpath & "nodefile.txt" For Random Shared As nodelistfilenum Len = Len(
nodelistfile)

            'get first record from nodelistfile or list of nodes
            Get nodelistfilenum, nodelistposition, nodelistfile

            'do until nodelistfile is at a end
            Do Until EOF(nodelistfilenum)

                nodefilenum = FreeFile

                'open file specified in nodelistfile
                Open defaultpath & Trim(nodelistfile.node) & ".txt" For Random Shared As node
filenum Len = Len(nodefile)

                nodeposition = 1
                'get record in open file
                Get nodefilenum, nodeposition, nodefile

                'Open manual DDE links between the text box's and the database blocks in ques
tion
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "machname.A_C
V", devicenametext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "machtyp.A_CV
", devicetypetext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "machset.A_CV
", setuptimetext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "machdown.A_C
V", downtimetext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "machserv.A_C
V", serviceintervaltext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "filetyp.A_CV
", iffiletypetext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "commid.A_CV"
, communicationidtext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "commpar.A_CV
", communicationparameterstext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "partser.A_CV
", partservicetypetext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "feedsig.A_CV
", feedbacksignaltext)
                Call CommDDELink(nodelistfile.node & "." & "v" & nodeposition & "commport.A_C
```

```
V", communicationsporttext)

                'set contents of record in file to the textbox's
                devicenametext.Text = nodefile.devicename
                devicetypetext.Text = nodefile.devicetype
                setuptimetext.Text = nodefile.setuptime
                downtimetext.Text = nodefile.downtime
                serviceintervaltext.Text = nodefile.serviceinterval
                iffiletypetext.Text = nodefile.iffiletype
                communicationidtext.Text = nodefile.communicationid
                communicationparameterstext.Text = nodefile.communicationparameters
                partservicetypetext.Text = nodefile.partservicetype
                feedbacksignaltext.Text = nodefile.feedbacksignal
                communicationsporttext.Text = nodefile.communicationsport

                'Poke all data from the text box's to the database blocks concerned
                devicenametext.LinkPoke
                devicetypetext.LinkPoke
                setuptimetext.LinkPoke
                downtimetext.LinkPoke
                iffiletypetext.LinkPoke
                communicationidtext.LinkPoke
                communicationparameterstext.LinkPoke
                partservicetypetext.LinkPoke
                feedbacksignaltext.LinkPoke
                communicationsporttext.LinkPoke

                'close all DDE links for this node
                devicenametext.LinkMode = NONE
                devicetypetext.LinkMode = NONE
                setuptimetext.LinkMode = NONE
                downtimetext.LinkMode = NONE
                serviceintervaltext.LinkMode = NONE
                iffiletypetext.LinkMode = NONE
                communicationidtext.LinkMode = NONE
                communicationparameterstext.LinkMode = NONE
                partservicetypetext.LinkMode = NONE
                feedbacksignaltext.LinkMode = NONE
                communicationsporttext.LinkMode = NONE


                'close nodefile
                Close nodefilenum

                'Increment popsition to look at next record in nodelistfile file
                nodelistposition = nodelistposition + 1

                'get next record from nodelistfile or list of nodes
                Get nodelistfilenum, nodelistposition, nodelistfile

            Loop

        Else
            'do nothing
        End If
End If
Close nodelistfilenum
'Close DDE links if possible

End Sub
```

# Help main window

Help Guide provides general help on how to use this System Intergration Tool.

Select the Topic you would like to learn about, by clicking the mouse button whilst the mouse pointer is focused on the topic.

Topic:

- How does Virtual Factory Work?
- What is a typical Factory?
- How do I use Virtual Factory?

How to:

- Define the factory
- Define a Job
- Simulate the factory
- Monitor the factory
- Use the factory Reports Generator

HOW_TO.FRM - 1

```
Sub Start_Fix_Dmacs_Change ()

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)

'help selection

Select Case index

    Case 0
        help_VFwork.Show 1

    Case 1
        help_typicalfactory.Show 1

    Case 2
        help_howtouseVF.Show 1

    Case 3
        help_definefactory.Show 1

    Case 4
        help_defineajob.Show 1

    Case 5
        help_Simulatefactory.Show 1

    Case 6
        help_monitor.Show 1

    Case 7
        help_reports.Show 1

    Case Else
        'do nothing

    End Select

End Sub

Sub Label1_Click ()
    StartFix.Show 1
End Sub

Sub Label2_Click ()
    StopFix.Show 1
End Sub

Sub Label3_Click ()
    ChangeUserType.Show 1
End Sub

Sub Label4_Click ()
    Open_Cad.Show 1
End Sub

Sub Label5_Click ()
    Open_Cam.Show 1
End Sub

Sub mnuexit_Click ()
    Unload How_To
End Sub
```

# Job Definition help window

Defining a Job consists of the following steps:

1. Select Define Jobs from the main menu.

2. Enter the name of the new job and create the job, or open/modify an existing job.

3. Enter the Batch size, then either select Design to enter into CAD/CAM in order to produce the Machine files, or select define Tasks.

4. To define the Tasks which a Job consists of simply enter in all the information about each task in the job. The tasks for a particular job are performed sequentially. Thus if you first select a milling operation and then a Pressing operation, that is the order in which the tasks will be performed. Their is on limit to how many Tasks can be performed on any single job.

5. When leaving the Define Tasks screen you will be asked if you wish to place the job in the "To be processed Queue". This places the current job into a queue which it will stay in until it's due date equals that of the current date. Then that job will then be placed in the out going queue and will begin to be manufactured.

```
HELP_DE.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub mnuexit_Click ()
Unload help_defineajob

End Sub
```

# Factory Definition help window

Steps in setting up a Factory Using Virtual Factory:

1. To define a Factory, it is important to first know the setup simulated or real factory to be controlled .

2. Enter Enter the System Definition program from the main menu.

3. The Factory requires a number of different parts to be defined.

   A. The different cells or FixDmacs nodes on the network must be defined. This can be done by entering the Node name of all the Fixdmacs nodes to be found on the network into the list provided in the Define Nodes window.

   B. The layout of each of these nodes or cells then can be performed by using the Define Layout window. By selecting the Define Layout button and then by selecting the node to define it is possible to specify the layout of the Cell.

   C. Defining the layout of the Cell is a simple process, it consists of entering the specific information about the machines in each of the four possible positions in the cell.

4. Once the definitions for all the cells have been entered the factory can then be configured. This can be done by selecting the Configure button in the Factory definition Window. This will configure the virtual factory to the definitions provided.

```
HELP_DEF.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub mnuexit_Click ()
Unload help_definefactory

End Sub
```

# Using the Virtual Factory application help window

Using Virtual Factory Should consist of the following steps:

To gain further information on any of the Main Points Click on the desired topic.

1. The first step is to Define the system.

2. Once the Factory system has been specified. The Jobs to be processed must then be defined. This will specify what each job consists of and what will be required to manufacture it.

3. The factory can then be Simulated. This will enable the user to see how The factory defined will operate under the given conditions.

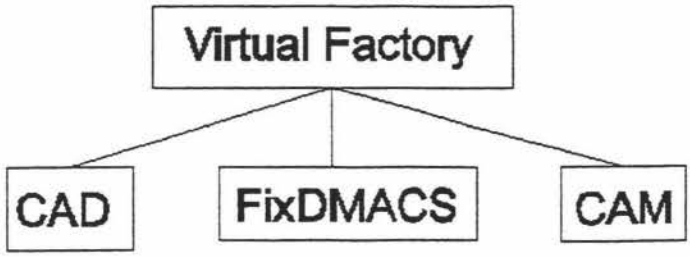
4. The Factory can also be Monitored in real time by the operator using the graphical interface.

5. The operator can then view the performace of the factory through using the Reports function. This enables the operator to guage how the system has operated.

```
HELP_HOW.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help1_Click (index As Integer)

Select Case index

    Case 0
        help_definefactory.Show 1

    Case 1
        help_definefactory.Show 1

    Case 2
        help_defineajob.Show 1

    Case 3
        help_simulatefactory.Show 1

    Case 4
        help_monitor.Show 1

    Case 5
        help_reports.Show 1

    Case Else
        'do nothing

    End Select

End Sub

Sub mnuexit_Click ()
Unload help_howtouseVF

End Sub
```

# Virtual Factory Structure help window

Virtual Factory is a combination of several software packages intergrated to give a single intergrated enviroment.




Virtual Factory utalises a Visual Basic front end program to provide the intergration between the different software packages. The front end program alows the user to define the factory enviroment as well as the jobs to be processed. This information is then passed on to the factory control software.

The Factory control software "FixDMACS" then controls the virutal factory as it has been defined by the user. This control package operates in real time and allows the user to see what is happening in the factory via it's visual interface. The real or physical factory is controlled jointly by FixDMACS and also by individual Programable Logic Controllers.

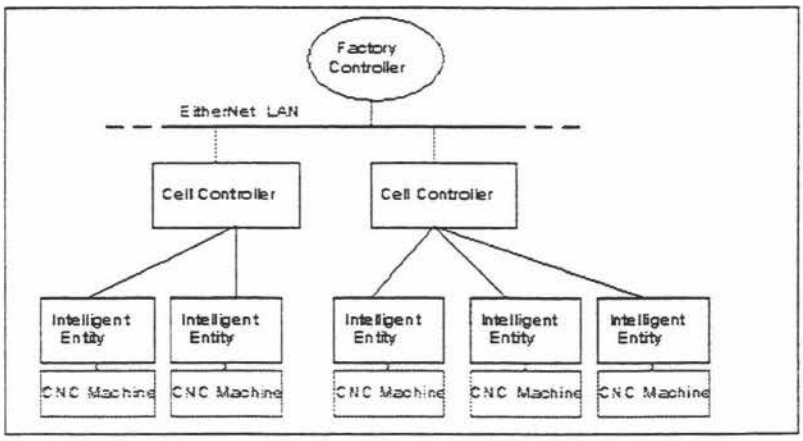
HELP_VFW.FRM - 1

```
Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub mnuexit_Click ()
Unload help_VFwork

End Sub
```

# Typical Factory Layout help window

**A typical Factory would consist of one Virtual Factory controller program like this one and several cell controllers. These cell controllers control the local cell by negotiating between the other cells in the factory and the factory controller itself for resources and jobs.**

**A Typical Factory Layout**



```
HELP_TYP.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub mnuexit_Click ()
Unload help_Typicalfactory

End Sub
```

# Task Definition help window

Select the option you require Help with

Topic:

- Job Task
  - Open
  - Modify
  - Delete
  - Create
- Assebly Task
  - Define

```
HOW_TOTA.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub helptaskdefinition_Click (index As Integer)

Select Case index

    Case 0
        MsgBox "Selecting OPEN, open's an existing Job Task file (*.iff) for modification."

    Case 1
        MsgBox "Selecting MODIFY, open's an existing Job Task file for modification."

    Case 2
        MsgBox "Selecting DELETE this deletes the specified in Job Name, or asks the user to
select a file to delete."

    Case 3
        MsgBox "Selecting CREATE, creates a new Job Task file (*.iff), with the name entered
in the Job Name box.  The file is created in the c:\data\progs directory."

    Case 4
        MsgBox "Selecting DEFINE, opens the define assembly program"

    Case Else

    End Select



End Sub

Sub mnuexit_Click ()
 Unload How_ToTaskDefinition
End Sub
```

# Node Definition help window

Select the option you require Help with

Topic:

- What is a node
- Add a new node
- Delete a Node

```
HOWTO_D.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)

Select Case index

    Case 0
        MsgBox "A Node or Cell is the name of a Fixdmacs Database, each database must communi
cate over the network and thus must have a seperate node name."

    Case 1
        MsgBox "To Add a new node simply enter the name of the new Fixdmacs Node into the spa
ce provided then select the Add button."

    Case 2
        MsgBox "To Delete a node select the node you wish to delete from the list provided th
en press the Delete Button."

    Case Else

End Select

End Sub

Sub mnuexit_Click ()
Unload howto_Definenodes
End Sub
```

# General Job help window

Select the option you require Help with

Topic:

- General
  - What is a Job?
  - What is a Task?
  - Modify a Job
  - Enter a New Job
- Task Control
  - Add a new Task
  - Delete a Task
  - Move Between Tasks Within a Job 
  - Save a Task

```
HOWTO_DE.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)

Select Case index

    Case Is = 0
        MsgBox "A JOB is a series of tasks which are undertaken by various CNC or other devic
es, these tasks/ processes operate on a part, with several parts going into producing a produ
ct."

    Case Is = 1
        MsgBox "A TASK is a process which is performed on a part.  The specifications for ind
ividual tasks are defined in this Job Definition screen."

    Case Is = 2
        MsgBox "To MODIFY a JOB open an existing Job by selecting OPEN on the previous Job De
finition Screen, then modify the individual tasks and save as you go."

    Case Is = 3
        MsgBox "To ADD a NEW task to the end of a job move to the end of the task using the u
p arrow and then press the up arrow one more time.  When you are presented with a blank scree
n this is a new record.  It is important NOT to save a black record.  The Last record must co
ntain the due Date and the final task check box must be checked if the job is to be processed
."

    Case Is = 4
        MsgBox "To DELETE a task, move to the task you wish to delete then press on the Delet
e button and that task will be erased."

    Case Is = 5
        MsgBox "To MOVE Between tasks use the up or down arrows on the move control in the bo
ttom left hand corner of the screen."

    Case Is = 6
        MsgBox "To ENTER a new JOB return to the Job Definition screen and select create on t
he Job Definition screen then continue on as normal."

    Case Is = 7
        MsgBox "To SAVE a task either move on to the next task and select Yes at the Request
to save prompt, or press the SAVE button under Task Control"

    Case Else

End Select


End Sub

Sub mnuexit_Click ()
Unload Howto_define_job
End Sub
```

# Task Definition help window

Select the option you require Help with

Topic

Design

Define Task

```
HOWTO_JO.FRM - 1

Option Explicit

Sub Label1_Click ()
MsgBox "Clicking on design takes you through to the part design Tools, such as CAD and CAM"
End Sub

Sub Label6_Click ()
MsgBox "Define Task, this option takes you into the modification or creation of the Job Task
information file, you must first enter a JOB BATCH SIZE"
End Sub

Sub mnuexit_Click ()
Unload Howto_Job_Management
End Sub
```

# Cell Definition help window

Select the option you require Help with

Topic:

- General Information
  - What is a Cell ?
  - What is the device number in the Cell ?
- How to
  - Modify an existing Cell definition
  - Define a new Cell
  - Move to the next Machine in the Cell

HOWTO_LA.FRM - 1

```
Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)

Select Case index

    Case 0
        MsgBox "A Cell is a area in a factory where at least one process is performed.  This
consists of no more than four machines all of which must perform a seperate process.  A cell
controller is a pysically seperate computer control program which controls a particular cell.
"

    Case 1
        MsgBox "The device number in the cell relates to its position and order within the ce
ll, this is shown in the 'Layout of Machines in Cell' diagram in the bottem left hand courner
 of the window."

    Case 2
        MsgBox "To modify an existing cell definition simply select the cell to define on the
 previous screen, then alter the appropriate values for the respective machines in the window
 provided."

    Case 3
        MsgBox "To define a new cell, return to main screen then select System Definition the
n press the Define Factory button and then select Define nodes/ Cells in factory.  Then type
in the new node name to the space provided and press the Insert button"

    Case 4
        MsgBox "To move to the next Machine in the cell, press the up arrow on the button in
the bottom center on the window"

    Case Else
        'do nothing

    End Select

End Sub

Sub mnuexit_Click ()
Unload Howto_layoutofcell
End Sub
```

# Machine Definition help window

Select the option you require Help with

Topic:

- Important Notes
- To Add a New Machine/ Device
- To Delete an unused machine/ Device
- What is the index
- How to leave without saving

```
HOWTO_MA.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)

Select Case index

    Case Is = 0
       MsgBox "When adding new devices it is important to use a unique index for each device.
 Each device must have an index. When a new device is added at this point, other parts of thi
s intergration package must also be updated. Pressing on the back button saves changes to the
 machine list, this change is perminent.  A new machine name can only be upto 14 characters l
ong and it's index can only be upto 5 digits"

    Case Is = 1
        MsgBox "To ADD a new device enter in the device name an it's unique index number and
press the add button"

    Case Is = 2
        MsgBox "To DELETE an unused machine from the machine file list, simply double click o
n the machine to delete form the list. IMPORTANT this change is not saved until the Back butt
on is pressed, to leave without saving use the exit choice from the File manu"

    Case Is = 3
        MsgBox "The index is a unique code which this application uses to identify each diffe
rent machine, it can be upto three digits"

    Case Is = 4
        MsgBox "To leave without saving, use the Exit choice from the File menu"

    Case Else

End Select

End Sub

Sub mnuexit_Click ()
Unload Howto_Machine_definition
End Sub
```

# Monitor Factory help window

Select the option you require Help with

Topic:

Monitor Factory

Monitor Queue

Monitor Jobs to be Manufactured

```
HOWTO_SY.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)

Select Case index

    Case 0
        MsgBox "To Monitor Factor press the Factory monitor button, this will bring up the fa
ctory summary screen."

    Case 1
        MsgBox "To monitor the Queue press the monitor Queue button, this will show the Queue
 status form.  The Queue status form displays the number of in the Queue and what the last jo
b that was used was."

    Case 2
        MsgBox "To monitor Jobs to be manufactured press the Jobs button, this will show the
jobs currently waiting for the date to equal their date so they can be placed in the queue to
 be manufactured."

    Case Else

    End Select

End Sub

Sub mnuexit_Click ()
Unload howto_System_Monitor
End Sub
```

# Queue application help window

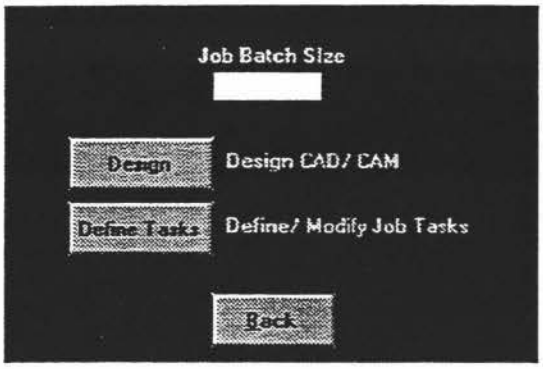
Select the option you require Help with

Topic:

- What is Job IF File Name List
- What is Due Date List
- Force Job to Queue
- Delete Job from Queue
- What is Current Date

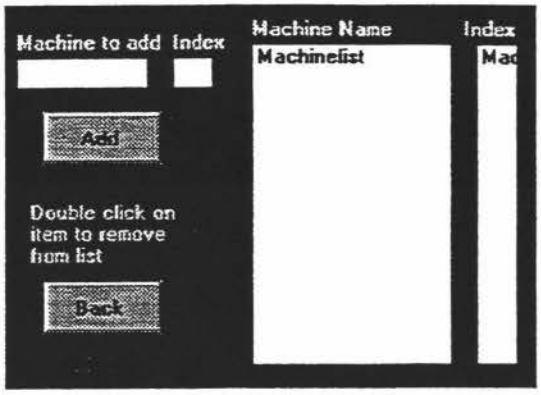
```
HOWTO_MO.FRM - 1

Option Explicit

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub help_Click (index As Integer)
Select Case index

    Case 0
        MsgBox "This is a list of all the Jobs that have been defined and are waiting for the
 date to equal their start date"

    Case 1
        MsgBox "This is a list of the due dates of the corisponding jobs in the IF File list"

    Case 2
        MsgBox "This forces the selected job in the IF Job File list onto the top of the out
going queue"

    Case 3
        MsgBox "This deletes the selected job from the IF Job File list"

    Case 4
        MsgBox "The current date is todays date"

    Case Else

    End Select

End Sub

Sub mnuexit_Click ()
Unload Howto_Monitor_Jobswaiting
End Sub
```

## Open Job file window



```
JOB_MANA.FRM - 1

Option Explicit

'**This form is designed to let the user select wether to design a job using CAD, CAM or to d
efine the job information


Sub Back_Click ()
'return to previous screen

Unload job_management

End Sub

Sub Define_Tasks_Click ()

Load Queue_Form

' check to see if a batch size has been entered
If jobbatchsize.Text = "" Then
    MsgBox "Please enter job batch size", 48, "Define Tasks"
    Exit Sub
    End If

define_Job.Show 1
Unload job_management

End Sub

Sub Design_Click ()
' Goto CAD/ CAM
design_Job.Show 1

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'This reads the name of the job into the title bar of this form
job_management.Caption = Job_Definition_Screen.Jobname.Text

' If file is not new then get the batch size form the file
On Error Resume Next
If isfilenew <> newfile Then

    Call readfromfile(recordname)
    jobbatchsize.Text = recordname.noofcycles
    'This is just to stop this from fireing again when clearalltaskinfo runs
    End If

End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()
Unload job_management
End Sub

Sub mnuhowto_Click ()
Howto_Job_Management.Show 1
End Sub

Sub mnuusinfhelp_Click ()
usehelp.Show 1
End Sub
```

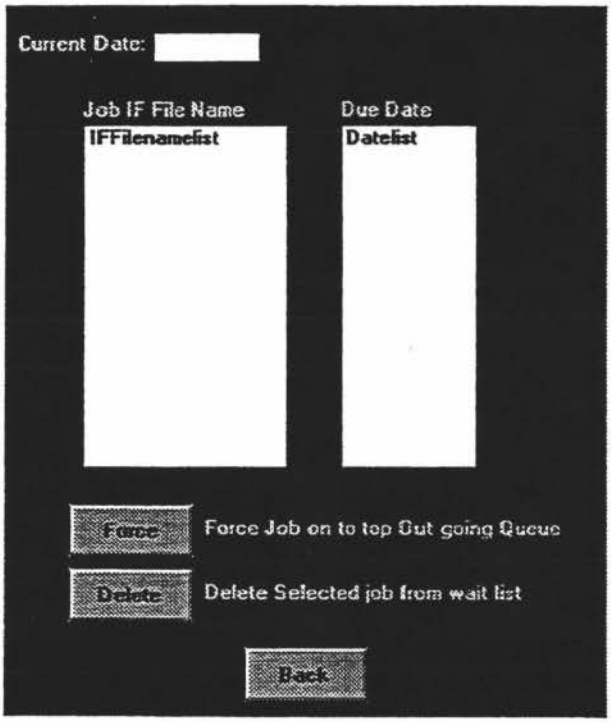
# Machine Edit window



```
MACHINE_.FRM - 1

Option Explicit

'**This form is designed to allow the user to alter the machines avaliable to be used
'**The user enters a machine name and a index number in to the appropriate txt box's
'**Each index must be unique
'**This is then stored in a file call c:\data\prog_man\machinlt.mch as two fields in a record


Sub Add_Tolist_Click ()
' Add from text box into list
Machinelist.AddItem machine.Text
Machineindexlist.AddItem machineindex.Text

End Sub

Sub Back_Click ()
'This is designed to delete the old machinlt file and replace it with the contents of the lis
t
'This removes any old records

Dim i As Integer
Dim recordlen As Integer
Dim Machinenames As machinerecord

' Kill old file
Close
Kill defaultpath & "Machinlt.mch"

On Error Resume Next
' Create new file specs
recordlen = Len(recordname)
machinefilenum = FreeFile

' Opening file
Open defaultpath & "machinlt.mch" For Random Shared As machinefilenum Len = recordlen

' This puts the information from the list box into a the file
For i = 1 To Machinelist.ListCount
    Machinenames.Machinename = Machinelist.List(0)
    Machinenames.currentindex = Machineindexlist.List(0)
    Put machinefilenum, i, Machinenames
    Machinelist.RemoveItem 0
    Machineindexlist.RemoveItem 0
    Next

Close machinefilenum

Unload Machine_Definition


End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

' Load from file and input into list

Dim Machinenames As machinerecord
Dim recordlen As Integer
Dim ind As Integer
Dim machineposition As Integer

'On Error Resume Next

' Create new file specs
recordlen = Len(recordname)
machinefilenum = FreeFile
machineposition = 1

' Opening file
Open defaultpath & "Machinlt.mch" For Random Shared As machinefilenum Len = recordlen

If Not EOF(machinefilenum) Then
ind = 1
    ' This is to place contents of file into list on screen
```

```
    Get machinefilenum, ind, Machinenames
    Do Until EOF(machinefilenum) Or Machinenames.Machinename = ""
        Machinelist.AddItem Machinenames.Machinename
        Machineindexlist.AddItem CStr(Machinenames.currentindex)
        ind = ind + 1
        Get machinefilenum, ind, Machinenames
     Loop

    End If
Close machinefilenum
End Sub

Sub Machineindexlist_DblClick ()
Dim i As Integer

For i = 0 To (Machineindexlist.ListCount - 2)
        If Machineindexlist.Selected(i) Then
            Machineindexlist.RemoveItem i
        End If
    Next i

End Sub

Sub Machinelist_DblClick ()
Dim i As Integer

For i = 0 To (Machinelist.ListCount - 2)
        If Machinelist.Selected(i) Then
            Machinelist.RemoveItem i
        End If
    Next i

End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()
Unload Machine_Definition
End Sub

Sub mnuhowto_Click ()
Howto_Machine_definition.Show 1
End Sub

Sub mnuusing_Click ()
usehelp.Show 1
End Sub
```

# Monitor Job Release Queue Window



```
MONITOR_.FRM - 1

Option Explicit

Dim datefilenum1 As Integer

Sub deletefromdatefile (record As daterecord, position As Integer)

'Pass in the record structure from which to delete from

' This deletes a record from the file

    record.date = ""
    record.filename = ""
    Put filenum, position, record

End Sub

Sub loadjoblist (record As daterecord)

' This procedure is designed to load all the records from the datefile.txt file and place the
 date and file name into a list
' this will allow he user to view jobs waiting to be manufactured

'Declare variables

Dim recordlen1 As Integer
Dim ind As Integer
Dim position As Integer


' Create new file specs
recordlen1 = Len(record)
filenum = FreeFile
position = 1

' Opening file machine list file
Open defaultpath & "datefile.txt" For Random Shared As filenum Len = recordlen1

'Do until reachs end of file
If Not EOF(filenum) Then

    ' This is to place contents of file into list on screen
    For ind = position To LOF(filenum) / (Len(record))
        Get filenum, ind, record

        ' If the record is not empty then add to list else continue
        If Trim(record.filename) <> "" Then
            IFFilenamelist.AddItem record.filename
            Datelist.AddItem record.date
            End If
        Next
End If
Close filenum

End Sub

Sub writetoscreen (record As jobrecord)
' This procedure writes the data in the fields of jobrecord type passsed into this procedure
' to the input fields of the next Queue_form

' To let user Know when a job has been precessed

Queue_Form!NJobNameText.text = record.jobname
Queue_Form!NIFFileNameText.text = datefile.filename
Queue_Form!NCycleTimeText.text = record.estcycletime
Queue_Form!NSetuptimeText.text = record.setuptime
Queue_Form!NMachineTypeText.text = record.Machinetype
Queue_Form!NNoofPartsText.text = record.noofcycles
Queue_Form!NToolsRequiredText.text = record.toolsrequired
Queue_Form.NLoadunloadtimetext.text = record.Estloadunloadtime
Beep

End Sub

Sub Back_Click ()
Unload Monitor_jobswaiting

End Sub
```

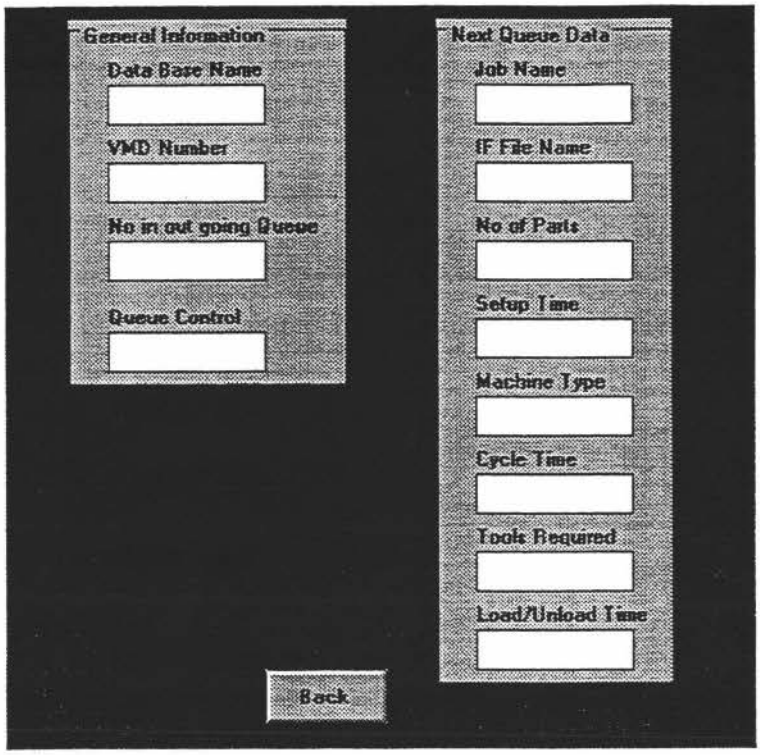
```
MONITOR_.FRM - 2

Sub Delete_Click ()

'This procedure deletes the selected item from the IFFilelist listbox from the file and from
the list


If IFFilenamelist.ListIndex <> -1 Then

    Dim position As Integer
    Dim datefile As daterecord
    Dim temp1 As Integer

    filenum = FreeFile
    position = 1

    ' Opening file machine list file
    Open defaultpath & "datefile.txt" For Random Shared As filenum Len = Len(datefile)

    'Loop until found end of record or record to be deleted
    Do

    Get filenum, position, datefile

        If Trim$(IFFilenamelist.List(IFFilenamelist.ListIndex)) = Trim$(datefile.filename) Th
en
            ' The record to delete has been found
            'Delete record from file
            Call deletefromdatefile(datefile, position)
            'Delete items from list using index of selected item
            temp1 = IFFilenamelist.ListIndex
            IFFilenamelist.RemoveItem IFFilenamelist.ListIndex
            Datelist.RemoveItem temp1
            Exit Do
          Else
            position = position + 1
            End If

    Loop Until Trim(IFFilenamelist.List(IFFilenamelist.ListIndex)) = Trim(datefile.filename)
Or EOF(filenum)

    Close filenum
Else
    'No item selected
    MsgBox "Please Select item from list to be deleted"

End If

End Sub

Sub Force_Click ()

'This procedure scans the datefile.txt file for any records with the record.filename = to tha
t selected on the IFFilename list
'if this occurs then the job is pushed onto the front of the queue and that record is delete
from the datefile.txt file.

On Error Resume Next

'*Declartions for datefile program
Dim datefile As daterecord
Dim dateposition As Integer
Dim recordlen1 As Integer
Dim datefilenum As Integer

datefilenum = FreeFile

' Opening file date file program
Open defaultpath & "datefile.txt" For Random Shared As datefilenum Len = Len(datefile)
dateposition = 1


'*Delcarations for IF file transfer between the file specified in the date file and the queue
ing program
Dim tempjobrecord As jobrecord
Dim jobposition As Integer
Dim jobrecordlen As Integer
Dim jobdatefilenum As Integer
Dim temp1 As Integer
```

```
' Search file until until reached end of file
Do Until EOF(datefilenum)
    'Get date from first record in file
    Get datefilenum, dateposition, datefile

    ' If the name of job in the file is the same as the name selected on the list then
    If Trim(datefile.filename) = Trim(IFFilenamelist.List(IFFilenamelist.ListIndex)) Then
        'Set file control variable
        jobdatefilenum = FreeFile

        'open IF file specified in datefile.filename field.
        Open defaultpath & Trim(datefile.filename) For Random Shared As jobdatefilenum Len =
Len(tempjobrecord)
        jobposition = 1

        'get first task in file record
        Get jobdatefilenum, jobposition, tempjobrecord
        'set the force field in the record to 1
        temprecord.force = 1
        Put jobdatefilenum, jobposition, tempjobrecord

        'load information from file into next queue
        Call writetoscreen(tempjobrecord)
        'Push onto end of the next queue
        Queue_Form!QueControlText.text = 0
        Queue_Form!QueControlText.text = 4

        'deletes record from datefile
        Call deletefromdatefile(datefile, dateposition)
        Close jobdatefilenum

        'Remove items from list on form
         temp1 = IFFilenamelist.ListIndex
         IFFilenamelist.RemoveItem IFFilenamelist.ListIndex
         Datelist.RemoveItem temp1


    End If

    dateposition = dateposition + 1
    Get datefilenum, dateposition, datefile

Loop

Close datefilenum


End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994
'
'This form is to display all the jobs in the datefile.txt file
'this form displays these jobs in list

currentdate.text = Date

' open file and load all into list
Dim tempdatefile As daterecord
Call loadjoblist(tempdatefile)

End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()
Unload Monitor_jobswaiting
End Sub

Sub mnuhowto_Click ()
Howto_Monitor_JobsWaiting.Show 1
End Sub
```

MONITOR_.FRM - 4

```
Sub mnuusing_Click ()
usehelp.Show 1
End Sub
```

# Queue application window



```
NEXTQUE.FRM - 1

Option Explicit

Option Base 1                'Begins the arrays at 1

Dim Nextqueue(100) As queue

Const Close_DDE = 5, Open_DDE = 1, NONE = 0, Link_Manual = 2, Link_Automatic = 1
  'Constants for 1 and 5

Dim Nstartofqueue As Integer
Dim NEndofqueue As Integer

'For the Pop and Put function and procedures both Job_Name and IF_Filename
'require the name of the Queue to be entered.  For the rest of the Queue name
'is set for "thisqueue".
'The Pop function returns the value specified by the name
'The Put procedure passes in the postion in the queue and the value to be inserted
' at present the position to insert into the queue can only be at the end
' or at the start of the queue.  A 0 indicates the begining of the queue whilst
' a 1 indicates the end of the queue.

Dim LinkItem As String
Dim DDETextName As Variant

Dim VMDName As String

Const Finishofqueue = 100
Const Beginofqueue = 1


Sub CommDDELink (LinkItem, DDETextName As TextBox)

'This procedure is designed to run once on startup to set up a DDE link with fixdmacs
'The Block name is passed in as Linkitem and the DDE Text box name is passed in as
' DDETextName


Dim firsttime As Double                          'This is the variable for the timer function

If DDETextName.LinkMode = NONE Then              'If not already open
        On Error Resume Next
        DDETextName.LinkTopic = "DMDDE|DATA"     'set topic
        DDETextName.LinkItem = LinkItem          'set link item, this is passed in
        DDETextName.LinkMode = Link_Manual    'link mode is automatic
        End If


On Error GoTo Errorhandel

Exit Sub

Errorhandel:
    MsgBox " An Error Has Occured in Setting up the DDE Links! "
    Exit Sub
End Sub

Sub Back_Click ()
Queue_Form.Hide
End Sub

Sub Databasetext_Change ()
' This text block contains the name of the node that this queue will be communicating with

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994
'
''
'**This application is a Queueing program
'This queueing program excepts data from the datefile.txt file passed to it from the timer ev
ent this is then placed into the queue to wait for bidding

'DDE links are used to transfer information between Fix Dmacs and This application
```

NEXTQUE.FRM - 2

```
'On text box has an automatic link, this controls all of the functioning of the program
'The other text box's contain either setup information or Data to be stored or retrieved from
 the queue

'This application links with VMD 1 in the factory node

On Error Resume Next

'Initialise queue pointers for the array
Nstartofqueue = Beginofqueue
NEndofqueue = Beginofqueue

' InputBox's to enter database name and vmd number oh year oh yaa ya ya ya yaaaaa forget abou
t the last one get yourself another!
If databasetext.text = "" And VMDNumbertext.text = "" Then
    databasetext.text = InputBox("Please enter the Name of this Node", "Node Definition", "CI
M2")
    VMDNumbertext.text = 1
    End If


'Set up the automatic link for the transfer of control information between Fix and VB

If quecontroltext.LinkMode = NONE Then              'If not already open
        quecontroltext.LinkTopic = "DMDDE|DATA"     'set topic
        quecontroltext.LinkItem = databasetext.text & "." & "V" & VMDNumbertext.text & "qcont
rol.A_CV"           'set link item, this is passed in
        quecontroltext.LinkMode = Link_Automatic    'link mode is automatic
        End If

 'Set up Manual Links for all of the text box's
 ' Next Queue
 Call CommDDELink(databasetext.text & "." & "Bidpartno.A_CV", NNoofpartstext)
 Call CommDDELink(databasetext.text & "." & "bidsetupm.A_CV", Nsetuptimetext)
 Call CommDDELink(databasetext.text & "." & "bidmachtyp.A_CV", NMachinetypetext)
 Call CommDDELink(databasetext.text & "." & "bidcyctm.A_CV", NCycletimetext)
 Call CommDDELink(databasetext.text & "." & "bidtooltyp.A_CV", NToolsRequiredtext)
 Call CommDDELink(databasetext.text & "." & "bidjob.A_CV", Njobnametext)
 Call CommDDELink(databasetext.text & "." & "bidiffile.A_CV", NIFFilenametext)
 Call CommDDELink(databasetext.text & "." & "vlnxtqueno.A_CV", NoinNqueuetext)
 Call CommDDELink(databasetext.text & "." & "bidloadun.A_CV", Nloadunloadtimetext)


'stop the number going to 100
NoinNqueuetext.text = 0

End Sub

Sub mnuabout_Click ()
About.Show 1
End Sub

Sub mnuexit_Click ()

'This Quits this application
Queue_Form.Hide

End Sub

Sub NoinNqueuetext_Change ()

'Poke number in queue when this value changes
On Error Resume Next

NoinNqueuetext.LinkPoke

End Sub

Sub QueControlText_Change ()
' This is the Information File control block
'
' 1 Pops the values from the next queue to the appropriate text Box's
' 2 This Requests information from Fixdmacs to the Next Queue
' 3 Pushes the value from the appropriate text box into the next queue to the end of the queu
e
' 4 Pushes the value from the appropriate text box into the next queue at the start of the qu
eue

On Error Resume Next            'This stops the DDE transfer from tripping the error and stop
```

```
NEXTQUE.FRM - 3

ping the program


Select Case quecontroltext.text

    Case "A" To "Z", "", "a" To "z", "!" To "/", ":" To "@", "[" To "`", "{" To "~", " " To "
                "
        'Anyt other char that may be input by accident

    Case Is = 1
        'Pop from Next queue

        If Nstartofqueue = NEndofqueue Then
            'Queue is empty
            Exit Sub

        End If

        Nstartofqueue = Nstartofqueue + 1

        If Nstartofqueue > Finishofqueue Then
                'Start of queue is at end of queue wrap around
                Nstartofqueue = Beginofqueue
                End If

        Njobnametext.text = Nextqueue(Nstartofqueue).job_Name
        NIFFilenametext.text = Nextqueue(Nstartofqueue).IF_Filename
        NNoofpartstext.text = Nextqueue(Nstartofqueue).No_ofparts
        Nsetuptimetext.text = Nextqueue(Nstartofqueue).setup_time
        NMachinetypetext.text = Nextqueue(Nstartofqueue).machine_type
        NCycletimetext.text = Nextqueue(Nstartofqueue).Cycle_Time
        NToolsRequiredtext.text = Nextqueue(Nstartofqueue).tools_required
        Nloadunloadtimetext.text = Nextqueue(Nstartofqueue).loadunload_Time

        'This pokes the information popped from the top of the queue to the factory controlle
r
        Njobnametext.LinkPoke
        NIFFilenametext.LinkPoke
        NNoofpartstext.LinkPoke
        Nsetuptimetext.LinkPoke
        NMachinetypetext.LinkPoke
        NCycletimetext.LinkPoke
        NToolsRequiredtext.LinkPoke
        Nloadunloadtimetext.LinkPoke

    Case Is = 2

        'Requesting Information from Fixdmacs
        Njobnametext.LinkRequest
        NIFFilenametext.LinkRequest
        NNoofpartstext.LinkRequest
        Nsetuptimetext.LinkRequest
        NMachinetypetext.LinkRequest
        NCycletimetext.LinkRequest
        NToolsRequiredtext.LinkRequest
        Nloadunloadtimetext.LinkRequest

    Case Is = 3
        'Push to end of Next queue

        If NEndofqueue + 1 = Nstartofqueue Then
            'Queue full
            GoTo fullmessage
            Exit Sub

        End If

        NEndofqueue = NEndofqueue + 1

        If NEndofqueue > Finishofqueue Then
            'If endofqueue at the end of the queue then wrap around
            NEndofqueue = Beginofqueue
         End If

        Nextqueue(NEndofqueue).job_Name = Njobnametext.text
        Nextqueue(NEndofqueue).IF_Filename = NIFFilenametext.text
        Nextqueue(NEndofqueue).No_ofparts = NNoofpartstext.text
        Nextqueue(NEndofqueue).setup_time = Nsetuptimetext.text
```

```
        Nextqueue(NEndofqueue).machine_type = NMachinetypetext.text
        Nextqueue(NEndofqueue).Cycle_Time = NCycletimetext.text
        Nextqueue(NEndofqueue).tools_required = NToolsRequiredtext.text
        Nextqueue(NEndofqueue).loadunload_Time = Nloadunloadtimetext.text

    Case Is = 4
        'Push to start of Next queue

        If NEndofqueue = Nstartofqueue - 1 Then
            ' Queue full
            GoTo fullmessage
            Exit Sub

        ElseIf Nstartofqueue < Beginofqueue Then
            Nstartofqueue = Finishofqueue

        End If

        Nextqueue(Nstartofqueue).job_Name = Njobnametext.text
        Nextqueue(Nstartofqueue).IF_Filename = NIFFilenametext.text
        Nextqueue(Nstartofqueue).No_ofparts = NNoofpartstext.text
        Nextqueue(Nstartofqueue).setup_time = Nsetuptimetext.text
        Nextqueue(Nstartofqueue).machine_type = NMachinetypetext.text
        Nextqueue(Nstartofqueue).Cycle_Time = NCycletimetext.text
        Nextqueue(Nstartofqueue).tools_required = NToolsRequiredtext.text
        Nextqueue(Nstartofqueue).loadunload_Time = Nloadunloadtimetext.text
        Nstartofqueue = Nstartofqueue - 1

    Case Else
        'This occurs when nothing else does

End Select


' Calculating the number in each queue

If NEndofqueue = Nstartofqueue Then
   NoinNqueuetext.text = 0

ElseIf NEndofqueue > Nstartofqueue Then
    NoinNqueuetext.text = NEndofqueue - Nstartofqueue

Else
    NoinNqueuetext.text = (100 - Nstartofqueue) + NEndofqueue

End If

Exit Sub

fullmessage:
    Dim msg As String
    msg = VMDNumbertext.text & " Next queue full"
    MsgBox msg, MB_ICONSTOP, "Queue Status"
    Exit Sub
End Sub



Sub VMDNumberText_Change ()

' This text block contains the name of the VMD number this queue will communicate with

End Sub
```

There are two ways to open the CAD package:

1/ From the Task Definition Screen, click on the CAD button, or press ALT + C

2/ From the main menu, choose the "Window" option, then the "Task Definition" option, and finally the "CAD" option.

OPEN_CAD.FRM - 1

```
Sub Command1_Click ()
    Unload Open_Cad
End Sub

Sub Command2_Click ()
    Unload Open_Cad
    ChangeUserType.Show 1
End Sub

Sub Command3_Click ()
    Unload Open_Cad
    Open_Cam.Show 1
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub
```

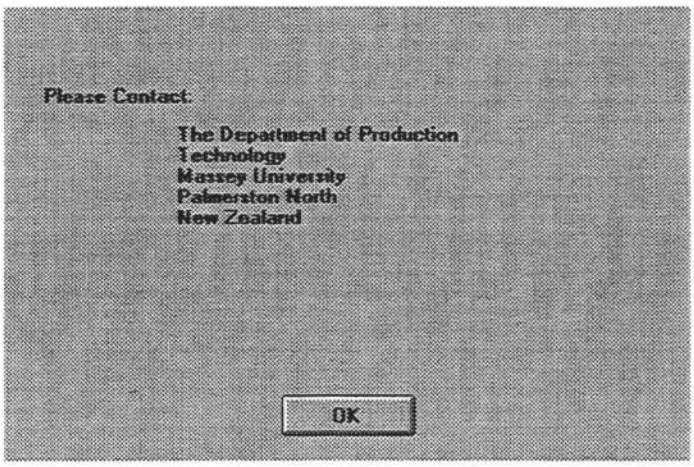
There are two ways to open the CAM package:

1/ From the Task Definition Screen, click on the CAM button, or press ALT + A

2/ From the main menu, choose the "Window" option, then the "Task Definition" option, and finally the "CAM" option.

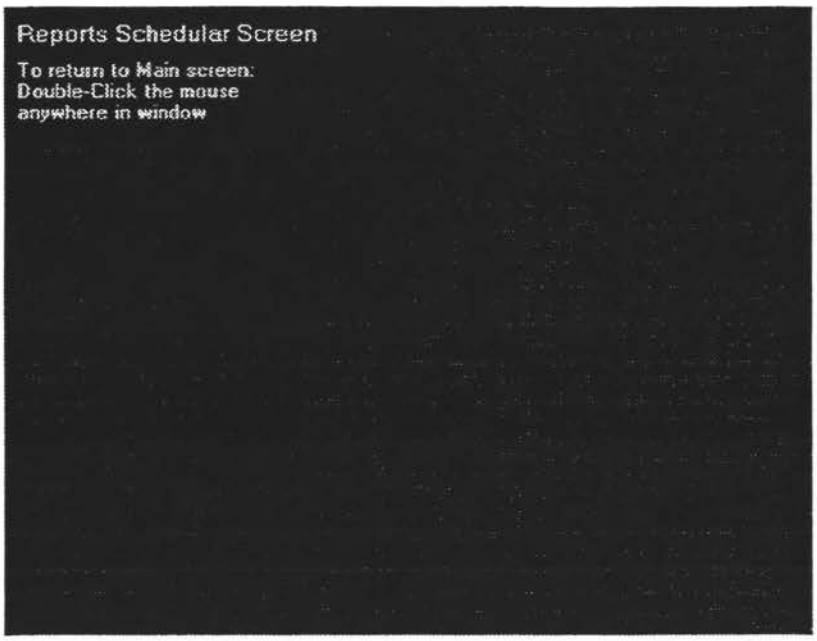
OPEN_CAM.FRM - 1

```
Sub Command1_Click ()
    Unload Open_Cam
End Sub

Sub Command2_Click ()
    Unload Open_Cam
    Open_Cad.Show 1
End Sub

Sub Command3_Click ()
    Unload Open_Cam
    StartFix.Show 1
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub
```


Please Contact:
The Department of Production
Technology
Massey University
Palmerston North
New Zealand
OK

PRODUCT_.FRM - 1

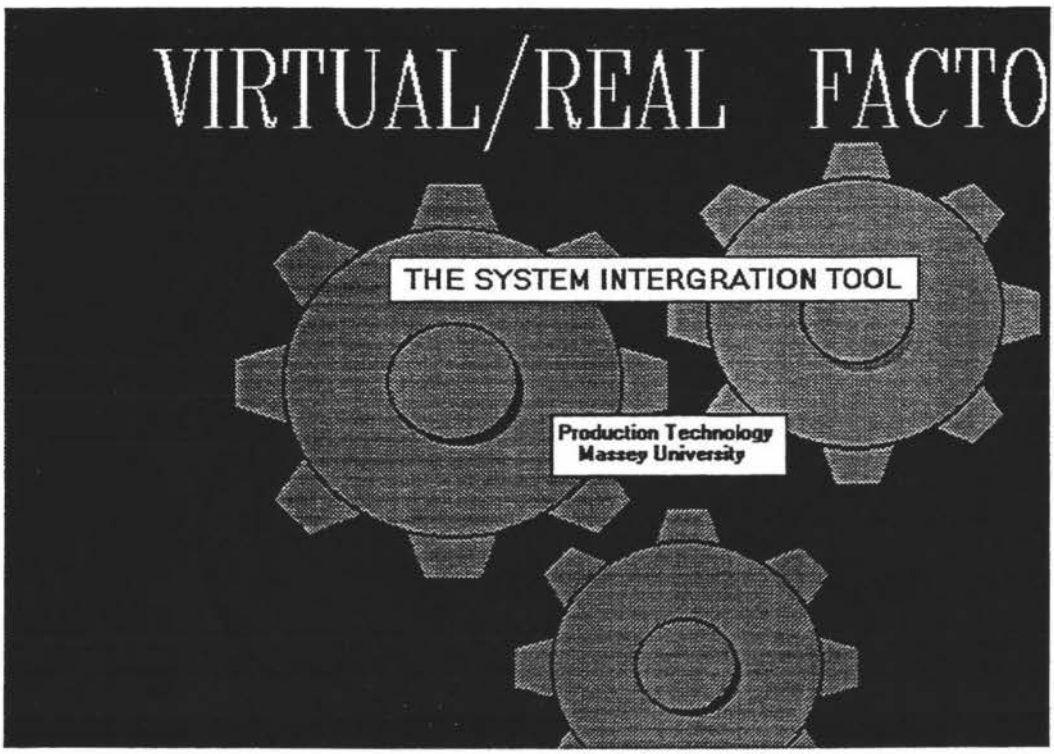
```
Sub Command1_Click ()
    Unload Product_Support
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub
```


Reports Schedular Screen
To return to Main screen:
Double-Click the mouse
anywhere in window

SIMULATE.FRM - 1

```
Sub Form_DblClick ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994
    Unload Simulate_Screen
End Sub
```

## About the Virtual Factory application



```
START_FO.FRM - 1

Option Explicit

Sub Picture1_Click ()
main_screen.Show 0
End Sub

Sub Start_Form_Change ()
main_screen.Show 0
End Sub

Sub Text2_Change ()
main_screen.Show 0
End Sub

Sub Text3_Change ()
main_screen.Show 0
End Sub

Sub Text5_Change ()
main_screen.Show 0
End Sub

Sub Form_Click ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

main_screen.Show 0
Unload start_form

End Sub

Sub Form_DblClick ()
main_screen.Show 0
Unload start_form

End Sub

Sub Form_KeyPress (KeyAscii As Integer)
main_screen.Show 0
Unload start_form

End Sub

Sub Image1_Click ()
main_screen.Show 0
Unload start_form

End Sub

Sub Label1_Click ()
main_screen.Show 0
Unload start_form

End Sub

Sub Text1_Click ()
main_screen.Show 0
Unload start_form

End Sub

Sub Text6_Change ()
main_screen.Show 0
End Sub
```

Close << >>

There are three methods for starting FixDmacs:

1/ Choose Start FixDmacs from the file menu, by clicking the mouse on the command, or by using the keyboard and pressing ALT+F then S

2/ FixDmacs can be started independantly of this application by double-clicking on the FIX DMACS Startup icon in the Fix Dmacs program group.

3/ If FIX DMACS is not running, you may encounter a message box asking if you would like to start it. Choose yes, or an abusive message will be displayed.

STARTFIX.FRM - 1

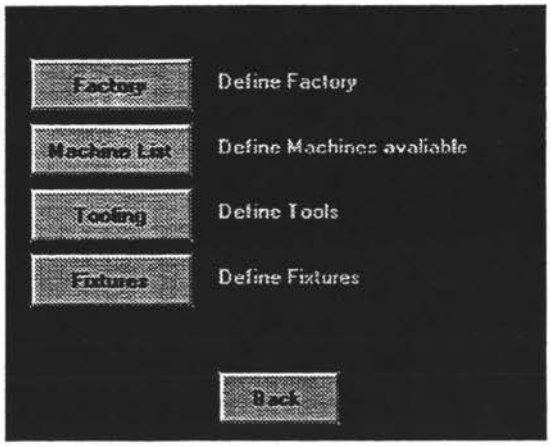
```
Sub Exit_Help_Click ()
    Unload StartFix
    'Unload How_To
End Sub

Sub Command1_Click ()
    Unload StartFix
End Sub

Sub Command2_Click ()
    Unload StartFix
    Open_Cam.Show 1
End Sub

Sub Command3_Click ()
    Unload StartFix
    StopFix.Show 1
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub
```


Return to the main screen for the System Integration Tool. Under the file menu, choose the "Close FixDmacs" option, by clicking on it with the mouse. Alternatively, press ALT + F, then C.

STOPFIX.FRM - 1

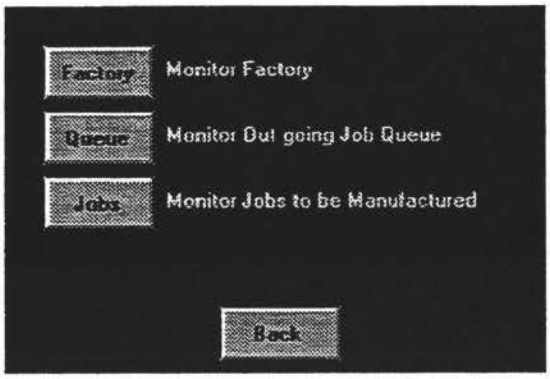
```
Sub Command1_Click ()
    Unload StopFix
End Sub

Sub Command2_Click ()
    Unload StopFix
    StartFix.Show 1
End Sub

Sub Command3_Click ()
    Unload StopFix
    ChangeUserType.Show 1
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub
```

# Factory Definition main menu window



```
SYSTEM_D.FRM - 1

Option Explicit

Sub Back_Click ()
Unload system_Definition
End Sub

Sub Define_Factory_Click ()
' first go into a screen which defines the nodes avaliable in the system, thus this can be us
ed to DDE to different cells and to
' rotate bidding

Factory_Definition.Show 1

End Sub

Sub Define_MachineList_Click ()
Machine_Definition.Show 1

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub mnuabout_Click ()
about.Show 1

End Sub

Sub mnuexit_Click ()
Unload system_Definition

End Sub

Sub mnuhowto_Click ()
 MsgBox "To define any of the Topics selected below press the Releted Button"
End Sub

Sub mnuusing_Click ()
usehelp.Show 1

End Sub
```

# Monitoring Factory main menu window



```
SYSTEM_M.FRM - 1

Option Explicit

Sub Back_Click ()
Unload system_monitor
End Sub

Sub factory_Click ()

Dim Response, x, title                    'Variable to see if Fix is running
                                 'Check if Fix is running...
If FixRunning = "No" Then        'If not, inform user
    Response = MsgBox("FIX DMACS is not currently running. Do you wish to start it now?", 4,
title)
    If Response = IDYES Then
        x = Shell("C:\WDMACS\FIX.EXE", 1)
        FixRunning = "Yes"

    Else
        MsgBox "Fixdmacs is not operating", 32, title
    End If
'ElseIf FixRunning = "Yes" Then 'If it is, start task
   ' x = Shell("C:\WDMACS\VIEW.EXE", 1)
   ' SendKeys "%", True
    'SendKeys "{ENTER}", True
   ' SendKeys "{ENTER}", True
   ' SendKeys "factory.odf", True
   ' SendKeys "{ENTER}", True
   SendKeys "{ENTER}", True
End If

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub jobs_Click ()
Monitor_Jobswaiting.Show 1
End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()
Unload system_monitor
End Sub

Sub mnuhowto_Click ()
howto_System_monitor.Show 1
End Sub

Sub mnuusinghelp_Click ()
usehelp.Show 1
End Sub

Sub Queue_Click ()

'*Start queue control form
Load Queue_Form

Queue_Form.Show 1

End Sub
```

To find out how to do a paticular task, choose "How To" in the help menu, and a list of topics will appear.

The Topics which provide help appear in green and when pressed will provied further information on that topic.

This is shown in all help windows with the following message.

Select the option you require Help with

USEHELP.FRM - 1

```
Sub Command1_Click ()
    Unload UseHelp
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub
```

MAINPROG.BAS - 1

```
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'Define a global variable to store status of user.
'This value will be Peasant, Supervisor, or Manager.

Global User

'Define a variable to determine whether or not FIX is running
'Values will be "Yes" or "No"

Global fixrunning, DDERunning, DBaseRunning

Global cell 'current cell no

'***This sets the factory controller in the factory***
Const factory_Controller = "cim2"


'Variables for every cell position:
Global cell1TL, cell1TR, cell1BL, cell1BR
Global cell2TL, cell2TR, cell2BL, cell2BR
Global cell3TL, cell3TR, cell3BL, cell3BR
Global cell4TL, cell4TR, cell4BL, cell4BR
Global cell5TL, cell5TR, cell5BL, cell5BR
Global cell6TL, cell6TR, cell6BL, cell6BR
Global cell7TL, cell7TR, cell7BL, cell7BR
Global cell8TL, cell8TR, cell8BL, cell8BR
Global cell9TL, cell9TR, cell9BL, cell9BR

' ***Definitions for job and task selection***
'
' This type contains all the fields/ information required about a particular job

Type jobrecord

    jobname As String * 13
    currentindex As Integer
    Finaltask As Integer
    tasknumber As Integer
    Machinetype As Integer
    taskname As String * 13
    Estcycletime As Integer
    setuptime As Integer
    Estloadunloadtime As Integer
    Noofcycles As Integer
    Machinefilename As String * 13
    Geometricfilename As String * 13
    Toolsrequired As Integer
    Fixturesfilename As String * 13
    Status As Integer
    duedate As String * 8
    force As Integer

End Type

' Define job record type and global file management variables
Global Recordname As jobrecord  'Record structure
Global temprecord As jobrecord

' Status variables for file manipulation
Global filestatus As Integer
Global isfilenew As Integer
Global filepath As String
Global position As Integer
Global filenum As Integer

' Constants for file usage
Global Const filehasbeenopened = 1
Global Const filehasbeenclosed = 0
Global Const defaultpath = "r:\fix\"
Global Const newfile = 1
Global Const modifyfile = 2
'***End***
'
'
```

MAINPROG.BAS - 2

```
'***Definitions for machine selection file type and ascociated parts (*.mch) ***
'
' Type definitions
Type machinerecord
    Machinename As String * 14
    currentindex As Integer
End Type
'
' file manipulation variables
Global machinefilenum As Integer
'***End***
'
'
'***Definitions for Date record file type and related variables
'
'Type definition
Type daterecord
    Filename As String * 13
    Date As String * 8
End Type
'
'File manipulation variables
Global datefilenum As Integer
Global datefile As daterecord
'***End***
'
'
'***Definitions for node record type, this type contains all the information about the cells
in a particular node.
'
Type noderecord

    nodename As String * 10
    Devicename As String * 20
    Devicetype As Integer
    setuptime As Integer
    downtime As Integer
    serviceinterval As Integer
    IFFiletype As Integer
    CommunicationID As Integer
    communicationparameters As String * 40
    Partservicetype As String * 15
    feedbacksignal As String * 100
    communicationsport As Integer

End Type
'
'***END***
'

'***Definitions for node list record type
'
Type nodelistrecord
    node As String * 10

End Type
'***END***



Sub readfromfile (record As jobrecord)
'read from file at position into recordname
If position = 0 Then
    position = 1
    End If

Get filenum, position, record

End Sub

Sub savetofile (record As jobrecord)

Dim response As Integer

' save all information to file
response = MsgBox("This task has been modified do you wish to save it?", MB_YESNO, "Save Job"
)
    If response = IDYes Then
```

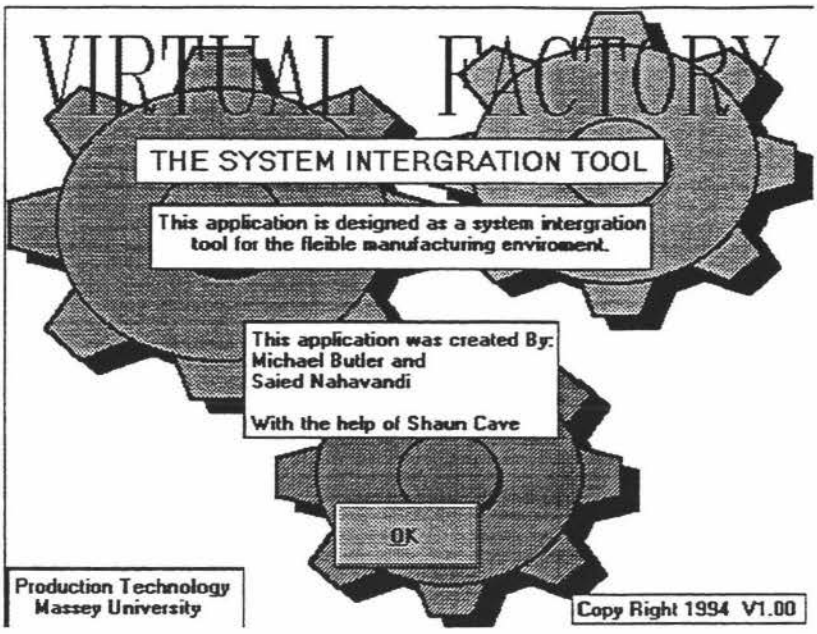
```
    ' save file
     Put #filenum, position, record
  End If
    ' No save file
 End Sub

Sub testfix ()

'This function is designed to warn users that Fixdmacs must be operating to perform some of t
he functions in this application

Dim response, response2 As Integer                    'Variable to see if Fix is running
                                'Check if Fix is running

If fixrunning = IDYes Then

    'This is fine
    Exit Sub

    Else
    'fixrunning = IDNo or something else

    response = MsgBox("Is FIX DMACS currently running.", 4, Title)

    If response = IDNo Then
        response = MsgBox("This operation requires Fixdmacs to be operating.  Do you wish to
continue.", 4, Title)

        If response2 = IDNo Then
            'end and pass out that this operation will not continue

            fixrunning = IDNo
            Exit Sub
            End If

         MsgBox "Before continuing on with this operation you must first Switch to program ma
nager and start Fixdmacs"

         ElseIf response = IDYes Then
            'then fix is running

            fixrunning = IDYes
         End If

    End If

End Sub
```

QUEUE1.BAS - 1

```
Option Explicit
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'**For queue program
'** I dont know why this works but it does?***
Type queue

    Job_Name As Variant
    IF_Filename As Variant
    No_ofparts As Variant
    Setup_Time As Variant
    Tools_Required As Variant
    Machine_Type As Variant
    Cycle_Time As Variant
    Loadunload_Time As Variant

End Type

Global Const factory_Controller = "cim2"


' MsgBox parameters
Global Const MB_OK = 0                  ' OK button only
Global Const MB_OKCANCEL = 1            ' OK and Cancel buttons
Global Const MB_ABORTRETRYIGNORE = 2    ' Abort, Retry, and Ignore buttons
Global Const MB_YESNOCANCEL = 3         ' Yes, No, and Cancel buttons
Global Const MB_YESNO = 4               ' Yes and No buttons
Global Const MB_RETRYCANCEL = 5         ' Retry and Cancel buttons

Global Const MB_ICONSTOP = 16           ' Critical message
Global Const MB_ICONQUESTION = 32       ' Warning query
Global Const MB_ICONEXCLAMATION = 48    ' Warning message
Global Const MB_ICONINFORMATION = 64    ' Information message

Global Const MB_APPLMODAL = 0           ' Application Modal Message Box
Global Const MB_DEFBUTTON1 = 0          ' First button is default
Global Const MB_DEFBUTTON2 = 256        ' Second button is default
Global Const MB_DEFBUTTON3 = 512        ' Third button is default
Global Const MB_SYSTEMMODAL = 4096      'System Modal

' MsgBox return values
Global Const IDOK = 1                   ' OK button pressed
Global Const IDCANCEL = 2               ' Cancel button pressed
Global Const IDABORT = 3                ' Abort button pressed
Global Const IDRETRY = 4                ' Retry button pressed
Global Const IDIGNORE = 5               ' Ignore button pressed
Global Const IDYES = 6                  ' Yes button pressed
Global Const IDNO = 7                   ' No button pressed


Function loadApplication (Appname, Path) As Variant

'Written by Michael Butler
'Department of Production Technology
'Massey University
'
'This function takes an application name and path then attempts to start it
'if any errors occur it aborts, returns loadapplication yes or no ie it has been loaded
'
'
'Improvement can be made here by testing if the application is open before it is opened using
 an API call
'

Dim response, x                                         'response from message box
                            'return from message box

    On Error GoTo ErrorHandel   'if an error occurs goto errorhandel
    x = Shell(Path & Appname)
    loadApplication = True

Exit Function

ErrorHandel:
```

```
    response = MsgBox("Is " & Appname & " currently running?", MB_YESNOCANCEL, "Application S
tatus")       'Ask if want to start application
    If response = IDYES Then
        loadApplication = True
    Else
        MsgBox "Error in opening application, this operation may not continue", 48, "Applicat
ion Status"
        loadApplication = False
    End If
    Exit Function

End Function
```

# Information about the Virtual Factory application



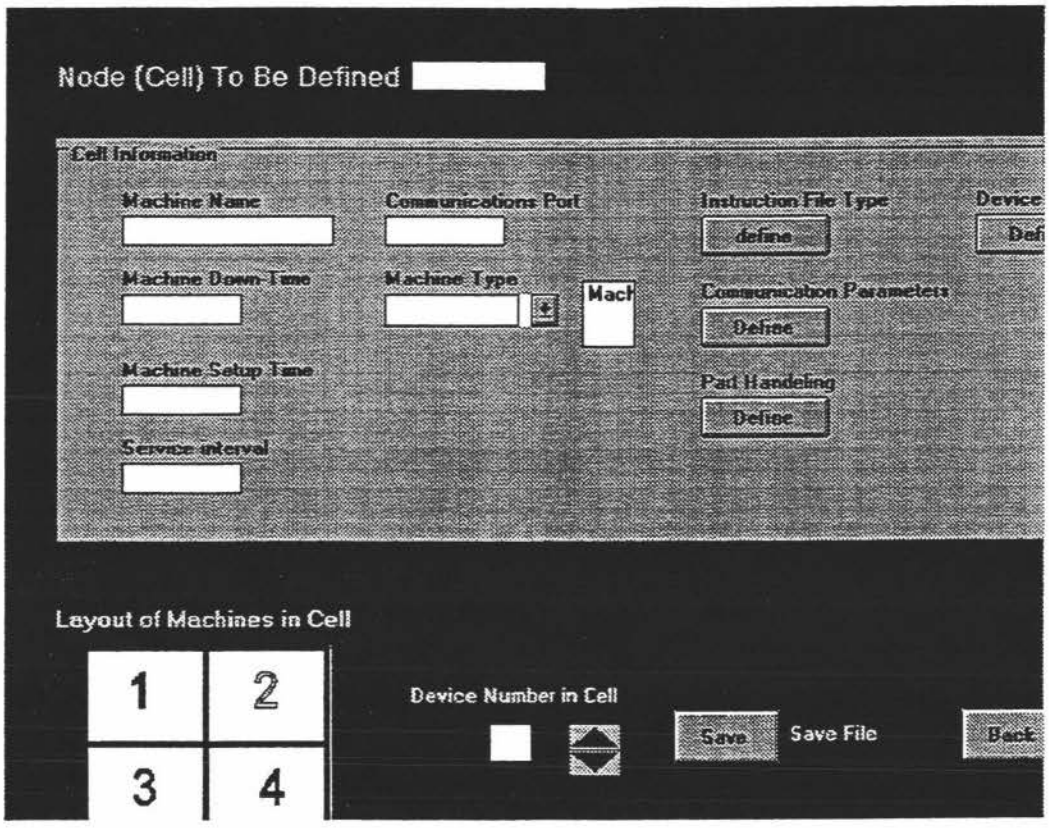
ABOUT.FRM - 1

```
Sub Command1_Click ()
    Unload about
End Sub

Sub Command2_Click ()
Unload about

End Sub
```

```
DEFINE_C.FRM - 2

    Call writetoscreen(record)

End If

Close filenum

Exit Sub

errorhandle6:
    If Err = 68 Then
        MsgBox "The Network drive " & defaultpath & " is not avaliable", MB_ICONSTOP, "Open n
odefile.txt"
        Else
            Resume Next
        End If
    Exit Sub

End Sub

Sub readfromscreen (record As noderecord)
'This procedure reads the value from the screen and places it in the variable/ record passed
in


'Read contents of screen
record.nodename = node.Text
record.devicename = Machine_name.Text
'This gets the index of the machine type in the list

If machine_type.ListIndex = -1 Then
    'If no item selected then scan through machine_type list until found item = text is combo
 box then find its index
    'and use the item in machine_index with the same index for the record.devicetype
    Dim i As Integer
    For i = 0 To machine_type.listcount
        If machine_type.Text = machine_type.List(i) And machine_type.List(i) <> "" Then
            record.devicetype = CInt(machine_index.List(i))
        End If
    Next i
    Else
    'else use the item with the same index as that selected in machine_type (list)
    record.devicetype = CInt(machine_index.List(machine_type.ListIndex))
    End If

record.setuptime = Setup_Time.Text
record.downtime = Down_Time.Text
record.serviceinterval = Service_Interval.Text
'IFFiletype, CommunicationID, Communication Parameters, Partservice type, feedback signal are
 not supported at the Fixdmacs leval and I have not included them in here
' as they are not supported at this level either
record.communicationsport = Communications_Port.Text


End Sub

Sub savetofile (record As noderecord, position As Integer)
'This procedure takes the contents of the screen and saves it to the file specified

Dim temprecord As noderecord

Call readfromscreen(temprecord)

record = temprecord

'This places the above record into the file nodefile.txt
Put filenum, position, record

End Sub

Sub writetoscreen (record As noderecord)
'This procedure writes the contents of the record passed to it to the screen

Machine_name.Text = record.devicename
Down_Time.Text = record.downtime
Setup_Time.Text = record.setuptime
Service_Interval.Text = record.serviceinterval
Communications_Port.Text = record.communicationsport

'This loop is to determine which item to put into the top of the machine type.text
```

```
DEFINE_C.FRM - 3

Dim i As Integer

For i = 0 To machine_index.listcount
    If record.devicetype = CInt(machine_index.List(i)) Then
        'Item from machinetype field in record = item in list
        machine_type.Text = machine_type.List(i)

        'Safe check in case it goes it to infinite loop
        ElseIf i > 100 Then
            Exit Sub
            End If
    Next i

End Sub

Sub Back_Click ()

'close open file
Close filenum

Unload Define_celllayout

End Sub

Sub Form_Load ()

'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'This form is to use the selected node given in previous form and to enter the information ab
out the given
'node into the appropriate text box's on screen. These are then saved to the file nodefile.tx
t for recall on factory startup.

'This puts the selected node from the previous screen into the node text box
node.Text = define_Layout!Nodelist.List(define_Layout!Nodelist.ListIndex)

'Start position at 1
position = 1

'Loads contents of record and outputs the first record to the screen
position = 1
Call loadnode(nodefile, position)

'This loads the machine type into the combo box
Call loadmachinetype

'Set the record position counter to 1
recordposition.Text = 1

End Sub

Sub Machine_Index_Click ()
'This list contains the index of the items from the machine_type list, each item in the machi
ne index list has a unique index
' thus this second invisable list has been used to store this index, thus when the user selec
ts a machine type it is not the machine type
' that is written to the file it is the index of the machine.
End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()
Unload Define_celllayout
End Sub

Sub mnuhowto_Click ()
Howto_Layoutofcell.Show 1
End Sub

Sub mnuusing_Click ()
usehelp.Show 1
End Sub
```

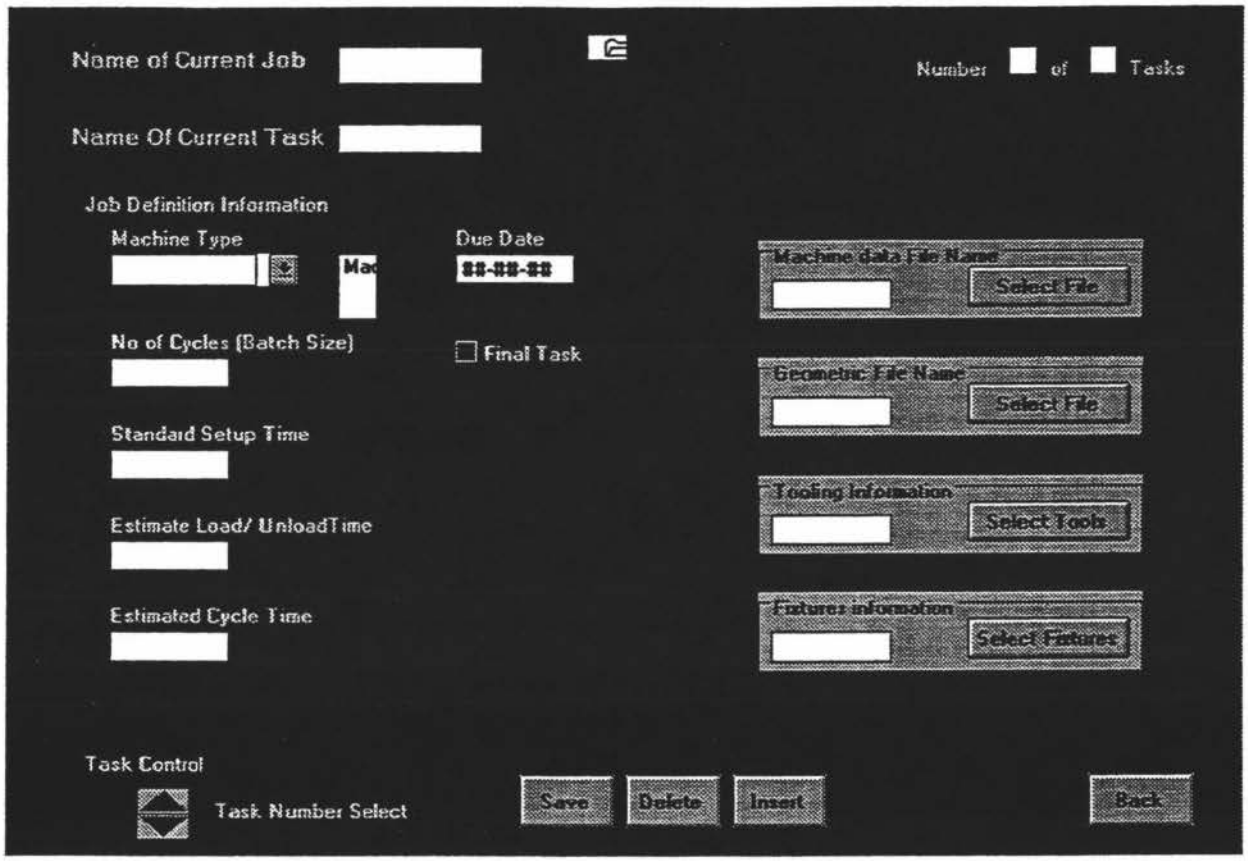
# Cell Definition main menu window



```
DEFINE_C.FRM - 1

Option Explicit

'Declarations for file use at the form level
Dim position As Integer
Dim nodefile As noderecord
Dim filenum As Integer

Sub clearscreen ()
' This procedure clears the contents of the screen

Machine_name.Text = ""
Down_Time.Text = ""
Setup_Time.Text = ""
Service_Interval.Text = ""
Communications_Port.Text = ""
machine_type.Text = ""

End Sub

Sub loadmachinetype ()

'Loading Machine items into machine list box

'Declare variables
Dim machinenames As machinerecord
Dim recordlen1 As Integer
Dim ind As Integer
Dim machineposition As Integer

'This is a procedure to load all the machine lists from the "Machinlt.mch" file into the comb
o box for selection by the user

' Create new file specs
recordlen1 = Len(recordname)
machinefilenum = FreeFile
machineposition = 1

' Opening file machine list file
Open defaultpath & "Machinlt.mch" For Random Shared As machinefilenum Len = recordlen1

'Do until reachs end of file
If Not EOF(machinefilenum) Then

    ' This is to place contents of file into list on screen
    For ind = machineposition To LOF(machinefilenum) / (Len(machinenames))
        Get machinefilenum, ind, machinenames
        machine_type.AddItem machinenames.Machinename
        machine_index.AddItem CStr(machinenames.currentindex)
        Next

        'Sets the text box to that of the first item in the list
        machine_type.Text = machine_type.List(0)
    End If

Close machinefilenum
End Sub

Sub loadnode (record As noderecord, position As Integer)

' This procedure is designed to load the current record specified from the nodefile.txt file
and in the node.text box
' then the contents of the current (specified by position) is then placed onto the screen

On Error GoTo errorhandle6

' Create new file specs
filenum = FreeFile

' Opening file machine list file
Open defaultpath & node.Text & ".txt" For Random Shared As filenum Len = Len(record)

'Get record from file
Get filenum, position, record

'Do until reachs end of file or the record read from the file on the disk is empty (record.de
vicename = "")
If Not EOF(filenum) Or record.devicename <> "" Then

    'writes values from the record in the file to the screen
```

```
DEFINE_C.FRM - 4

Sub recordselect_SpinDown ()
'This procedure will save the current contents of the screen to the file at the current posit
ion
' then decrement the counter and get the next record from the file then put it to the screen
On Error Resume Next

If position = 1 Then
    MsgBox "Unable to move to next Machine in Cell", MB_ICONINFORMATION, "Move down"
    Else
    'save the contents of the screen to disk
    Call savetofile(nodefile, position)

    'increment position
    position = position - 1
    recordposition.Text = CInt(recordposition.Text) - 1

    'clear screen
    Call clearscreen

    'load next record in file and send to screen
    Call loadnode(nodefile, position)
    End If

End Sub

Sub recordselect_SpinUp ()
'This procedure will save the current contents of the screen to the file at the current posit
ion
' then increment the counter and get the next record from the file then put it to the screen
On Error Resume Next

If position = 4 Then
    MsgBox "Unable to move to next Machine in Cell", MB_ICONINFORMATION, "Move down"
    Else
    'save the contents of the screen to disk
    Call readfromscreen(nodefile)
    Call savetofile(nodefile, position)

    'increment position
    position = position + 1
    recordposition.Text = CInt(recordposition.Text) + 1
    'clear screen
    Call clearscreen

    'load next record in file and send to screen
    Call loadnode(nodefile, position)
    End If

End Sub

Sub save_Click ()
'This will save the contents of the screen to the file at the current position

Call savetofile(nodefile, position)

End Sub
```

# Job Definition main menu



```
DEFINE_J.FRM - 1

Option Explicit

'**This form is designed to define the information required a job file
'**A job file is a file which will contain all the information about a job
'**that will enable it to be manufactured.
'**At this stage that information is obviously limited.
'**This Form takes the data entered in on the screen and writes it to a file which has alread
y been open in the form before last.
'**Related child forms are Select_geometricfiles, Select_Machinefiles, Select_toolsrequired a
nd most probably select_Fixturesfiles, Howto_define_Job
'**Related parent forms are Job_Definition_screen

'**This form also uses records of jobrecord type and two procedures called Readfromfile and S
avetofile, defined in module1

' Define job record type and file management variables
Const maxfilelen = 8


Sub clearalltaskinfor ()

' clear all of the text box's

current_taskname.Text = ""
machine_type.Text = ""
setup_time.Text = ""
machine_type.Text = ""
Estimated_loadtime.Text = ""
Estimated_cycletime.Text = ""
Machine_Filename.Text = ""
Geometric_filename.Text = ""
tools_requiredname.Text = ""
fixtures_filename.Text = ""
current_taskname.SelText = ""

End Sub

Sub deletefromfile (record As jobrecord)

' This deletes a record from a file
Dim ind As Integer

'This puts the next record in the position of the current record
For ind = position To LOF(filenum) / (Len(recordname)) - 1
    Get filenum, ind + 1, record
    record.tasknumber = CInt(record.tasknumber) - 1
    Put filenum, ind, record
    Next

' delete copy of last record, by reseting it's jobname and task number
Dim temprecord2 As jobrecord

    temprecord2.Jobname = ""
    temprecord2.taskname = ""
    Put filenum, CInt(define_job.Totaltasknumber.Text), temprecord2

'Alter the number of totaltasknumber to reflect the new total
Totaltasknumber.Text = CInt(Totaltasknumber.Text) - 1

End Sub

Sub loadmachinetype ()

'Loading Machine items into machine list box

'Declare variables
Dim machinenames As machinerecord
Dim recordlen1 As Integer
Dim ind As Integer
Dim machineposition As Integer

'This is a procedure to load all the machine lists from the "Machinlt.mch" file into the comb
o box for selection by the user

' Create new file specs
recordlen1 = Len(recordname)
MachineFilenum = FreeFile
machineposition = 1
```

DEFINE_J.FRM - 2

```
' Opening file machine list file
Open defaultpath & "Machinlt.mch" For Random Shared As MachineFilenum Len = recordlen1

'Do until reachs end of file
If Not EOF(MachineFilenum) Then

    ' This is to place contents of file into list on screen
    For ind = machineposition To LOF(MachineFilenum) / (Len(machinenames))
        Get MachineFilenum, ind, machinenames
        machine_type.AddItem machinenames.Machinename
        machine_index.AddItem CStr(machinenames.currentindex)
        Next
        'Sets the text box to that of the first item in the list
        machine_type.Text = machine_type.List(0)
    End If

Close MachineFilenum

End Sub

Sub readcurrenttaskinfor (record As jobrecord)

'This procedure reads all the inforation from the text box's on the form into the fields of t
he record of jobrecord type passed into this procedure

'If  nothing has been entered this generates an error this can be circumvented
On Error Resume Next

' get all info from screen put to temprecord
record.Jobname = Current_JobName.Text
record.finaltask = finaltask.Value
record.currentindex = position  'Sets current index in the record to position
record.tasknumber = CInt(Totaltasknumber.Text)

If machine_type.ListIndex = -1 Then
    'If no item selected then scan through machine_type list until found item = text is combo
 box then find its index
    'and use the item in machine_index with the same index for the record.devicetype
    Dim i As Integer
    For i = 0 To machine_type.ListCount
        If machine_type.Text = machine_type.List(i) Then
            record.Machinetype = CInt(machine_index.List(i))
        End If
        Next i
    Else
        'else use the item with the same index as that selected in machine_type (list)
        record.Machinetype = CInt(machine_index.List(machine_type.ListIndex))
    End If

record.taskname = current_taskname.Text
record.estcycletime = CInt(Estimated_cycletime.Text)
record.setuptime = CInt(setup_time.Text)
record.estloadunloadtime = CInt(Estimated_loadtime.Text)
record.noofcycles = CInt(No_ofcycles.Text)
record.machinefilename = Machine_Filename.Text
record.geometricfilename = Geometric_filename.Text
record.toolsrequired = CInt(tools_requiredname.Text)
record.fixturesfilename = fixtures_filename.Text
record.status = 0
record.duedate = due_date.Text

End Sub

Function testvarequivalent (temprecord1 As jobrecord, recordname1 As jobrecord) As Integer

'This procedure tests if the two variables of jobrecord type are the same as each other and p
asses out true if they are

If temprecord1.taskname = recordname1.taskname And temprecord1.estcycletime = recordname1.est
cycletime And temprecord1.duedate = recordname1.duedate And temprecord1.machinefilename = rec
ordname1.machinefilename And temprecord1.setuptime = recordname1.setuptime And temprecord1.es
tloadunloadtime = recordname1.estloadunloadtime And temprecord1.fixturesfilename = recordname
1.fixturesfilename And temprecord1.geometricfilename = recordname1.geometricfilename And temp
record1.toolsrequired = recordname1.toolsrequired And temprecord1.Machinetype = recordname1.M
achinetype And temprecord1.finaltask = recordname1.finaltask Then
    testvarequivalent = True
    Else
    testvarequivalent = False
```

```
    End If

End Function

Sub writecurrenttaskinfor (record As jobrecord)

'Write's contents of record of type jobrecord to the screen

current_taskname.Text = record.taskname
Estimated_cycletime.Text = record.estcycletime
setup_time.Text = record.setuptime
Estimated_loadtime.Text = record.estloadunloadtime
'This loop is to determine which item to put into the top of the machine type.text
Dim i As Integer

For i = 0 To (machine_index.ListCount - 1)
    If record.Machinetype = CInt(machine_index.List(i)) Then
        'Item from machinetype field in record = item in list
        machine_type.Text = machine_type.List(i)
        End If
    Next i

No_ofcycles.Text = record.noofcycles
finaltask.Value = record.finaltask
Machine_Filename.Text = record.machinefilename
Geometric_filename.Text = record.geometricfilename
tools_requiredname.Text = record.toolsrequired
fixtures_filename.Text = record.fixturesfilename
due_date.Text = record.duedate


End Sub

Sub writetodatefile (record As daterecord)
'
'**This procedure is designed to write the date and filename of a completed job to the file c
alled filedate.txt
' this is then a list of Job's waiting to be manufactured, and these Jobs are started dependi
ng on the date suppliedd
'
Dim dateposition As Integer
Dim recordlen As Integer

recordlen = Len(record)
datefilenum = FreeFile

' Opening file
Open defaultpath & "datefile.txt" For Random Shared As datefilenum Len = recordlen

dateposition = 1

Do Until EOF(datefilenum) Or Trim(record.filename) = ""

    Get datefilenum, dateposition, record
   ' If Trim(record.filename) = "" Or EOF(datefilenum) Then
        dateposition = dateposition + 1
        Get datefilenum, dateposition, record
        'Else
            'dateposition = dateposition + 1

      ' End If
Loop

'An empty record has been located write to this record
' or End of record has been reached and no empty records have been found, so write to end of
file
record.date = due_date.Text
record.filename = Current_JobName.Text
Put datefilenum, dateposition, record


Close datefilenum

End Sub

Sub Back_Click ()
'Returns back to previous form
'Saves data on screen
'updates datefile.txt file if requested
```

DEFINE_J.FRM - 4

```
Dim response As Integer

'Sets due date of job to due date in first record

Call readcurrenttaskinfor(temprecord)

response = MsgBox("Is  " & Current_JobName.Text & "  completed, and is " & due_date.Text & "
 the date to manufacture?", MB_YESNOCANCEL, "Define Job")

If response = IDYES Then

    ' This section writes the due date and file name to the filename.txt file for use by the
schedular algorithum
    ' Double check before writing to date record

    response = MsgBox("This Job is now completed and is ready to be manufactured, do you wish
 this Job to be placed into the To be processed Queue?", MB_YESNO, "Define Job")
        If response = IDYES Then
            position = 1
            Call readfromfile(temprecord)
            temprecord.duedate = due_date.Text
            temprecord.status = 0
            Call savetofile(temprecord)
            ' This writes to the datefile record
            Call writetodatefile(datefile)
            Else
            'continue
            End If

    ElseIf response = 2 Then 'If cancel is pressed
        Exit Sub

    Else
        Call readcurrenttaskinfor(temprecord)
        Call savetofile(temprecord)
    End If
'Close all open files
Close

Totaltasknumber.Text = CInt(Totaltasknumber.Text) + 1

Unload define_job

End Sub

Sub Command4_Click ()
Select_machinefilename.Show 1
End Sub

Sub Command5_Click ()
select_geometricfilename.Show 1
End Sub

Sub Command6_Click ()
select_Toolsrequired.Show 1
End Sub

Sub Current_TaskName_Change ()
'Let the contents of this text box become the title of this form
define_job.Caption = current_taskname.Text

End Sub

Sub Current_taskno_Change ()
' this equals current position
End Sub

Sub Delete_Click ()

'This procedure deletes the task at the current position a moves all records to fill the blan
k spot

Dim response, positionmark1, positionmark2 As Integer

positionmark1 = position

' this deletes the task at the current position
response = MsgBox("Do you wish to delete this task from the current Job", MB_YESNO, "Delete T
```

```
ask")

If response = IDYES Then
    ' Delete form file does all the work here
    Call deletefromfile(temprecord)
    ' This just reads the next file into the current position
    positionmark2 = position
    position = positionmark1
    Call readfromfile(recordname)
    Call writecurrenttaskinfor(recordname)
    position = positionmark2

    Else
    ' No delete file
    Exit Sub
    End If

End Sub

Sub Dir1_Change ()
' This drive control is here to get the current path for the use of opening files

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'Test if a new file if new then set current task no to 0 else set to 1
If isfilenew = newfile Then
    current_taskno.Text = 0
    Totaltasknumber.Text = 0
    Else
    current_taskno.Text = 1

    'This sets the total number of tasks to that of the length of the file
    Totaltasknumber.Text = (LOF(filenum) / (Len(recordname)))
    End If


'On opening read from *.iff file into screen if an error occurs assume no records in file exi
st.
On Error Resume Next

Call readfromfile(recordname)
Call writecurrenttaskinfor(recordname)

'Links job cycle time to job batch size
Current_JobName.Text = job_definition_screen.Jobname.Text
No_ofcycles.Text = job_management.Jobbatchsize.Text

' This loads machine information into combo box
Call loadmachinetype
Unload job_management

End Sub

Sub Insert_Click ()
MsgBox "This function does not yet work"
End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()
MsgBox "Please press 'BACK' to exit"

End Sub

Sub mnuhowto_Click ()
howto_define_job.Show 1

End Sub

Sub mnuusing_Click ()
```

```
usehelp.Show 1
End Sub

Sub save_Click ()
'This is a save function, and performs a save with no move to next task

'Save current screen inforamtion to file if a task and job name has been entered
If Current_JobName.Text = "" Then
    MsgBox "No Job Name has been Entered", 48, "Save Job to file"
    Exit Sub
    ElseIf current_taskname.Text = "" Then
         MsgBox "No task Name has been Entered", 48, "Save Job to file"
         Exit Sub
         End If

'Special case, at start of record
If CInt(current_taskno.Text) = 0 Then
    position = 1

End If

Call readcurrenttaskinfor(temprecord)
Call savetofile(temprecord)

current_taskno.Text = position

End Sub

Sub Select_TaskNo_SpinDown ()

' This is the move down control, on pressing it tests if current tasks has been modified, if
so it is saved then moves on to the next task in the down direction.

On Error GoTo errorhandel3

' If no task information entered then postion will be 0 or 1 then unable to move down to the
next task
If position = 0 Or position = 1 Then
    MsgBox "Unable to move to next task", 48, "Task Select"
    Exit Sub

   ElseIf position = CInt(current_taskno.Text) Then
    'In this case at position end of record moving down one, simply get record at current pos
ition - 1
    'Special case at end because their is a move then read unlike when the position is at any
 other point in the record
    position = position - 1
    Call readfromfile(recordname)
    Call writecurrenttaskinfor(recordname)

   Else
        ' Assume shifting to next task in record
        Call readcurrenttaskinfor(temprecord)
        Call readfromfile(recordname)
        If testvarequivalent(temprecord, recordname) = True Then
            ' If record in file is same as record in form then no save
            ' and load next task in record.
            position = position - 1
            Call readfromfile(recordname)
            Call writecurrenttaskinfor(recordname)
            Else
            ' Record is different from that in file, save? then load next file
            Call savetofile(temprecord)
            position = position - 1
            Call readfromfile(recordname)
            Call writecurrenttaskinfor(recordname)
            End If

        End If

' Let the current task number equal that of the position
current_taskno.Text = position

Exit Sub
errorhandel3:
    MsgBox "Unable to perform task", 48, "Task status"
    Exit Sub
End Sub
```

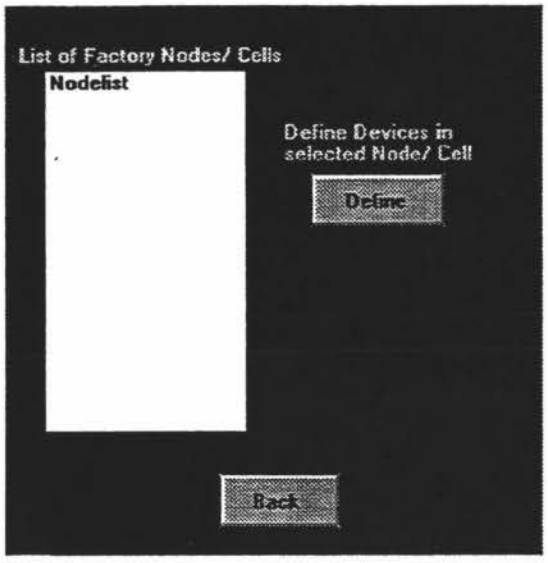
```
DEFINE_L.FRM - 1

Option Explicit

Sub loadnodelist (record As nodelistrecord)

' This procedure is designed to load all the records from the nodefile.txt file and place the
 node name into a list
' this will allow he user to view the nodes avaliable and also the node list will be avaliabl
e for the bidding process

'Declare variables

Dim recordlen1 As Integer
Dim ind As Integer
Dim position As Integer

On Error GoTo errorhandle6

' Create new file specs
recordlen1 = Len(record)
filenum = FreeFile
position = 1

' Opening file machine list file
Open defaultpath & "nodefile.txt" For Random Shared As filenum Len = recordlen1

'Do until reachs end of file
If Not EOF(filenum) Then

    ' This is to place contents of file into list on screen
    For ind = position To LOF(filenum) / (Len(record))
        Get filenum, ind, record

        ' If the record is not empty then add to list else continue
        If Trim(record.node) <> "" Then
            nodelist.AddItem record.node

            End If
        Next
End If
Close filenum
Exit Sub

errorhandle6:
    If Err = 68 Then
        MsgBox "The Network drive " & defaultpath & " is not avaliable", MB_ICONSTOP, "Open n
odefile.txt"
        Else
            Resume Next
        End If
    Exit Sub

End Sub

Sub Back_Click ()
Unload Define_Layout
End Sub

Sub Command1_Click ()
If nodelist.ListIndex = -1 Then
    MsgBox "Please click on the item from the list you wish to Define", MB_ICONINFORMATION, "
Delete Node"
    Else
        define_celllayout.Show 1
    End If

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'This form is to display all the nodes listed in the nodefile.txt file
'one of these nodes can then be selected to edit the respective layout file

' open file and load all into list
Dim nodelistfile As nodelistrecord
```

```
DEFINE_L.FRM - 2

Call loadnodelist(nodelistfile)

End Sub

Sub mnuabout_Click ()
about.Show 1

End Sub

Sub mnuexit_Click ()
Unload Define_Layout
End Sub

Sub mnuusing_Click ()
usehelp.Show 1

End Sub
```

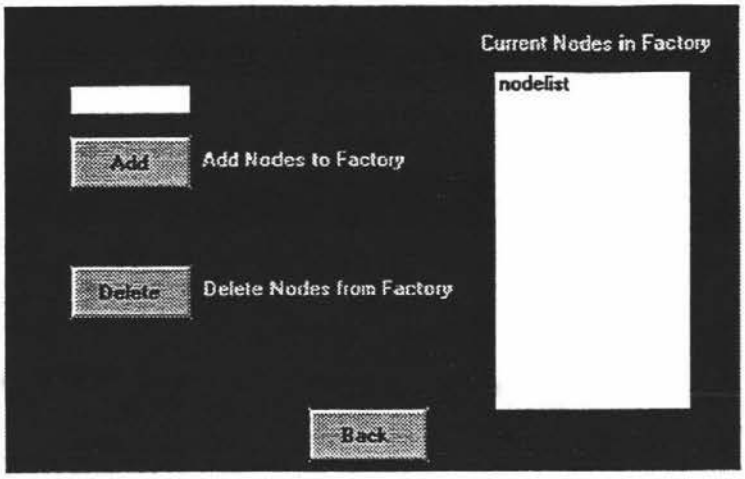
DEFINE_J.FRM - 7

```
Sub Select_TaskNo_SpinUp ()

' This is the move down control, on pressing it tests if current tasks has been modified, if
so it is saved then moves on to the next task in the up direction.
' This is the move down control

On Error GoTo errorhandel4

' If no task entered or final task then unable to move on to the next task
If current_taskname.Text = "" Or finaltask.Value = 1 Then
    MsgBox "Unable to move to next task", 48, "Task Select"
    Exit Sub

   ' If New job has been created then the first time around this will occur, special case
   ElseIf CInt(Totaltasknumber.Text) = 0 And EOF(filenum) Then
    position = 1
    Totaltasknumber.Text = 1
    Call readcurrenttaskinfor(temprecord)
    Call savetofile(temprecord)
    Call clearalltaskinfor
    ' No position move as position is already at 1, special case

   ' If end of record = current position, assume a new task is being added
   ElseIf CInt(Totaltasknumber.Text) = position Then
    ' Assume new job is being added
    'Update total position counter
    Totaltasknumber.Text = CInt(Totaltasknumber.Text) + 1
    position = position + 1
    Call readcurrenttaskinfor(temprecord)
    Call savetofile(temprecord)
    Call clearalltaskinfor

    Else
        ' Assume shifting to next task in record, but this task already exists
        Call readcurrenttaskinfor(temprecord)
        Call readfromfile(recordname)

        'Test if current task in record = that in the file ie has been changed
        If testvarequivalent(temprecord, recordname) = True Then
            ' If record in file is same as record in form then no save
            ' and load next task in record.
            position = position + 1
            Call readfromfile(recordname)
            Call writecurrenttaskinfor(recordname)
            Else
            ' Record is different from that in file, save? then load next file
            Call savetofile(temprecord)
            position = position + 1
            Call readfromfile(recordname)
            Call writecurrenttaskinfor(recordname)
            End If

        End If

' Let the current task number equal the position
current_taskno.Text = position

Exit Sub
errorhandel4:
    MsgBox "Unable to perform task", 48, "Task status"
    Exit Sub
End Sub
```

# Factory Node List main window



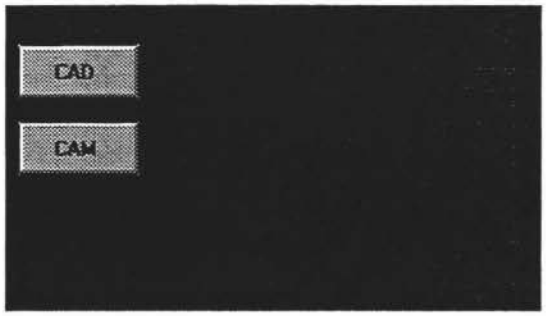
# Factory Node Definition main window



```
DEFINE_N.FRM - 1

Option Explicit

Sub loadnodelist (record As nodelistrecord)

' This procedure is designed to load all the records from the nodefile.txt file and place the
 node name into a list
' this will allow he user to view the nodes avaliable and also the node list will be avaliabl
e for the bidding process

'Declare variables

Dim recordlen1 As Integer
Dim ind As Integer
Dim position As Integer

On Error GoTo errorhandle6

' Create new file specs
recordlen1 = Len(record)
filenum = FreeFile
position = 1

' Opening file machine list file
Open defaultpath & "nodefile.txt" For Random Shared As filenum Len = recordlen1

'Do until reachs end of file
If Not EOF(filenum) Then

    ' This is to place contents of file into list on screen
    For ind = position To LOF(filenum) / (Len(record))
        Get filenum, ind, record

        ' If the record is not empty then add to list else continue
        If Trim(record.node) <> "" Then
            nodelist.AddItem record.node

            End If
        Next
End If
Close filenum
Exit Sub

errorhandle6:
    If Err = 68 Then
        MsgBox "The Network drive " & defaultpath & " is not avaliable", MB_ICONSTOP, "Open n
odefile.txt"
        Else
            Resume Next
        End If
    Exit Sub

End Sub

Sub Add_Click ()
' Add from text box into list

If New_Node.Text = "" Then
    MsgBox "Please enter the name of the Node you wish to add to the Node list", MB_ICONINFOR
MATION, "Add Node"
    Else
        nodelist.AddItem New_Node.Text
        New_Node.Text = ""

    End If

End Sub

Sub Back_Click ()
'This is designed to delete the old nodefile.txt file and replace it with the contents of the
 list
'This removes any old records

Dim count As Integer
Dim nodefilenum As Integer
Dim nodelistfile As nodelistrecord
On Error Resume Next

' Kill old file
Kill defaultpath & "nodefile.txt"
```

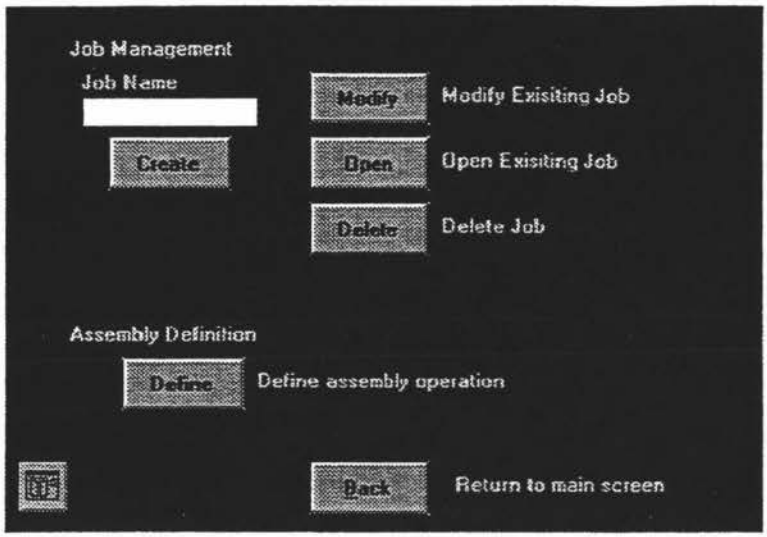
DEFINE_N.FRM - 2

```
' Create new file specs
nodefilenum = FreeFile

' Opening file
Open defaultpath & "nodefile.txt" For Random Shared As nodefilenum Len = Len(nodelistfile)

' This puts the information from the list box into a the file
For count = 1 To nodelist.ListCount
    nodelistfile.node = nodelist.List(0)
    Put nodefilenum, count, nodelistfile
    nodelist.RemoveItem 0
    Next

Close
Unload Define_Nodes

End Sub

Sub Delete_Node_Click ()

' This removes the selected item from the list

If nodelist.ListIndex = -1 Then
    MsgBox "Please click on the item from the list you wish to delete", MB_ICONINFORMATION, "
Delete Node"
    Else
        nodelist.RemoveItem (nodelist.ListIndex)
    End If

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994
'
'This form is to display all the nodes listed in the nodefile.txt file
'These nodes can then be added to a edited as need be

' open file and load all into list
Dim nodelistfile As nodelistrecord
Call loadnodelist(nodelistfile)

End Sub

Sub mnuabout_Click ()
About.Show 1
End Sub

Sub mnuexit_Click ()
Unload Define_Nodes
End Sub

Sub mnuhowto_Click ()
howto_definenodes.Show 1
End Sub

Sub mnuusing_Click ()
usehelp.Show 1
End Sub
```


CAD
CAM

```
DESIGN_J.FRM - 1

Option Explicit

Dim x As Variant

Sub Call_CAD_Click ()
 Dim x As Variant
 x = Shell("C:\autotrax\schedit\schedit.EXE", 1)

End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

End Sub

Sub mnuexit_Click ()
Unload design_job

End Sub
```

# Job Management window



```
DESIGN_S.FRM - 1


'**This form is for entering selecting the file to modify or define for either the job defini
tion or the assembly operation
Const maxfilelen = 8



Sub MaskedEdit1_ValidationError (InvalidText As String, StartPosition As Integer)
' In case of entering text to long
MsgBox "Job name invalid", MB_ICONEXCLAMATION, "Warning"
End Sub

Sub Assembly_Click ()
 define_assembly.Show 1
End Sub

Sub Back_Click ()

Close
Unload job_Definition_screen
End Sub

Sub Create_Click ()
' This creates a new file of the name specified in the text box

On Error GoTo errorhandle

' Check to see if able to create new file with name in text box
If jobname.Text = "" Then
    MsgBox "Please enter name of new job", 48, "Create new job"
    Exit Sub
    End If

'Checks to see if a file is already opened and thus the file control variables are already in
 use
If filestatus = filehasbeenopened Then
    MsgBox "Please close current file", 48, "Create new job"
    Exit Sub
    End If

' Create new file
recordlen = Len(recordname)
filenum = FreeFile

' Opening file
Open defaultpath & jobname.Text & ".iff" For Random Shared As filenum Len = recordlen
filestatus = filehasbeenopened
isfilenew = newfile

job_Management.Show 1
Unload job_Definition_screen

Exit Sub

errorhandle:
    If Err = 68 Then
            MsgBox "The Network drive " & defaultpath & " is not avaliable", MB_ICONSTOP, "Op
en nodefile.txt"
        Else
            MsgBox Error, 48, "File status"
        End If
    Exit Sub

End Sub

Sub Delete_Click ()

'This deletes a file as specified in the file name, and path supplied

Dim response As Integer

' Delete file, make sure all files are closed
Close

Dim Ansr, DelFile, Msg  ' Declare variables.
    On Error GoTo Errhandler     ' Set up error handler.
    'Asks for file to delete
    DelFile = UCase(InputBox("Enter file name you want deleted."))
    If Len(DelFile) Then     ' Check to see if file has any length.
```

```
        Ansr = MsgBox("Sure you want to delete " & DelFile & "?", 4)
        If Ansr = 6 Then     ' User chose "Yes."
            Msg = "Deleting " & DelFile & " from your disk."
            Kill DelFile     ' Delete file from disk.
        Else
            Msg = DelFile & " was not deleted."
        End If
    Else
        Msg = "You didn't enter a file name."
    End If
    MsgBox Msg  ' Display message.

Errhandler:
        If Err = 53 Then    ' Error 53 is "File not Found".

            Msg = "Sorry, the file you named could not be found."
        Else
            Msg = "Sorry, unable to delete file."
        End If
        Resume Next

End Sub

Sub Form_DblClick ()
    Unload job_Definition_screen
End Sub

Sub Form_Load ()
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

' Setting maximum length of jobname text box
jobname.MaxLength = maxfilelen
filestatus = filehasbeenclosed
End Sub

Sub mnuabout_Click ()
about.Show 1
End Sub

Sub mnuexit_Click ()

Unload job_Definition_screen

End Sub

Sub mnuhowto_Click ()
    How_ToTaskDefinition.Show 1
End Sub

Sub mnunusing_Click ()
Usehelp.Show 1
End Sub

Sub Modify_Click ()
'This opens a file for modification in the task definition screen

On Error GoTo errorhandle2

' If no name specified in text box then open file using commion dialog box else open using fi
le name supplied
If jobname.Text = "" Then

    ' Open job file, use commion dialog box
    'Sets the filters in the commion dialog box
    cmdialog1.Filter = "Job Files (*.iff)|*.iff|All files (*.*)|*.*"
    'Opens commion dialog box
    cmdialog1.Action = 1
    'gets name selected in dialog box
    jobname.Text = cmdialog1.Filetitle

    ' Create new file specs
    recordlen = Len(recordname)
    filenum = FreeFile

    ' Opening file as specified above
```

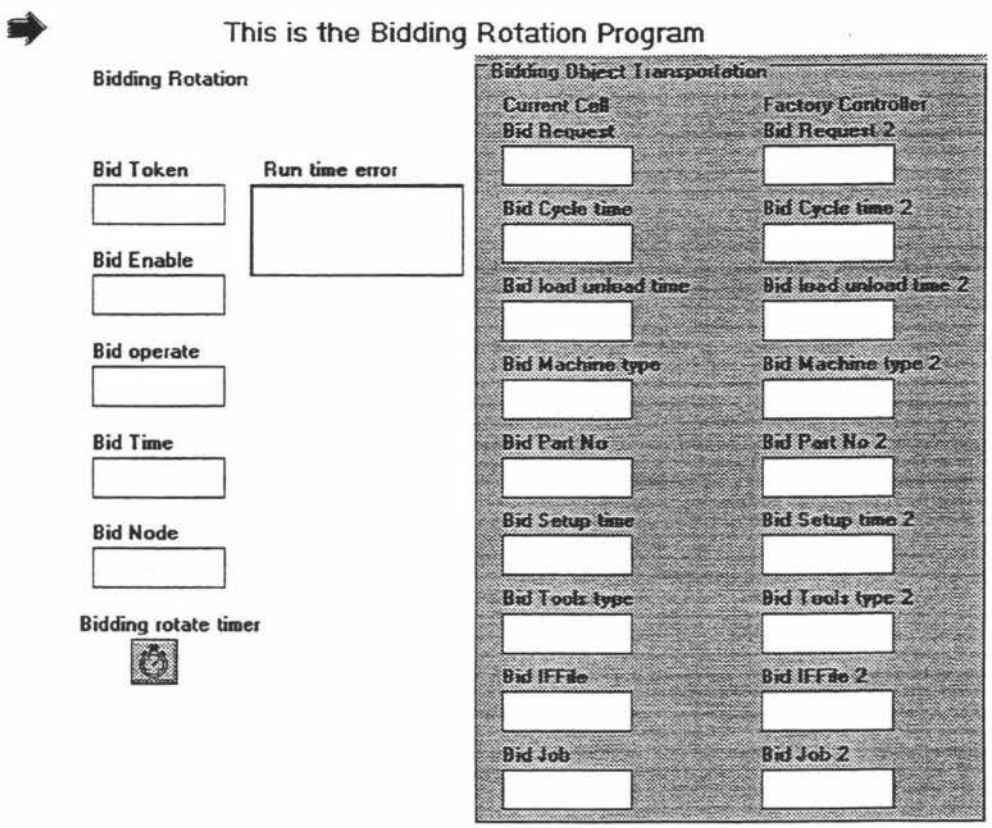
```
    Open cmdialog1.Filename For Random Shared As filenum Len = recordlen
    filestatus = filehasbeenopened
Else

    'Open file using job name text box in default path
    'Create or opens file
    recordlen = Len(recordname)
    filenum = FreeFile

    ' Opening file
    filepath = defaultpath ' sets path to the default path c:\data\pro_mang

    Open filepath & jobname.Text & ".iff" For Random Shared As filenum Len = recordlen
    filestatus = filehasbeenopened
    End If


job_Management.Show 1
Unload job_Definition_screen

Exit Sub
errorhandle2:
    MsgBox "File unable to be opened please retry", 48, "File status"
    Exit Sub

End Sub

Sub Open_Click ()
'This opens a file and works the same as the modify button

On Error GoTo errorhandle1

' Open job file, use commion dialog box

cmdialog1.Filter = "Job Files (*.iff)|*.iff|All files (*.*)|*.*"
cmdialog1.Action = 1
jobname.Text = cmdialog1.Filetitle

' Create new file specs
recordlen = Len(recordname)
filenum = FreeFile

' Opening file
Open cmdialog1.Filename For Random Shared As filenum Len = recordlen
filestatus = filehasbeenopened

job_Management.Show 1
Unload job_Definition_screen

Exit Sub
errorhandle1:
    MsgBox "File unable to be opened please retry", 48, "File status"
    Exit Sub

End Sub
```

# Appendix K

## Program Listing for Bid Rotation Visual Basic Program

This appendix contains the application listing and respective screen dumps, for the Bid Rotation application. The Bid Rotation application is the application which is responsible for the movement of the Bid Token around the factory's cells.

# Bid Rotation main application



```
BID_ROTA.FRM - 1

Option Explicit

Const Close_DDE = 5, Open_DDE = 1, none = 0, Link_Manual = 2, link_automatic = 1

'declare record type
Dim NodeListfile As nodelistrecord

'Declare variables for file manipulation
Dim position As Integer
Dim filenum As Integer

Dim counter As Integer

Sub CommDDELink (LinkItem, DDETextName As TextBox, link_type As Integer)

'This procedure is designed to run once on startup to set up a DDE link with fixdmacs
'The Block name is passed in as Linkitem and the DDE Text box name is passed in as
' DDETextName


On Error Resume Next
        DDETextName.LinkTopic = "DMDDE|DATA"     'set topic
        DDETextName.LinkItem = LinkItem          'set link item, this is passed in
        DDETextName.LinkMode = link_type         'sets up an link


End Sub


Sub resetoperate ()

'This procedure is designed to reset the bidoperate blocks before this bidrotate application
closes down
Close

' Create new file specs
filenum = FreeFile

' Opening file machine list file
Open defaultpath & "nodefile.txt" For Random Shared As filenum Len = Len(NodeListfile)

'Do until reachs end of file
If Not EOF(filenum) Then

    Do Until EOF(filenum)

        If Trim(NodeListfile.node) Like "???#" Then

           'if record is not empty then setup links for bidding enable
            Call CommDDELink(NodeListfile.node & ".bidenable.A_CV", bidenabletext, Link_Manua
l)

            ' set the bidenable to 0 to let the nodes know that a bodrcta application is clos
eing down
            bidenabletext.Text = 0
            bidenabletext.LinkPoke

            'Close DDE link to this DBB
            bidenabletext.LinkMode = none

            End If

            position = position + 1
            Get filenum, position, NodeListfile

            'safety trip
            If position = 20 Then
                MsgBox "File access error in nodefile.txt"
                End
                End If
        Loop
End If

'close file
Close filenum

End Sub
```

```
BID_ROTA.FRM - 2

Sub Bidrequest_Change ()

'This module is the control for the bidding object transportation
'If 1 is entered then the bidrequest is transported from the source node to the factory contr
oller
'If 2 is entered then the bidding objects are loaded from the source node into this applicati
on
'If 3 is entered then the data is transfered from the current source links to the factory lin
ks
'If 4 is entered then the bidding objects in the factory links are transported to the factory
 controller

'On Error Resume Next

'Select Case bidrequest.Text

 '   Case Is = 0
        'Reset bidrequest
  '      bidrequest2.Text = 0
    '
     'Case Is = 1
        'Transport the bidrequest to the fatory controller
      '   bidrequest2.Text = 1
       '  bidrequest2.LinkPoke
    .
    ' Case Is = 2
        'Request from source node all bidding objects except bid request
     '    bidcycletime.LinkRequest
      '   bidloadunloadtime.LinkRequest
       '  bidmachinetype.LinkRequest
        ' bidpartno.LinkRequest
         'bidsetuptime.LinkRequest
        ' bidtoolstype.LinkRequest
         'bidiffile.LinkRequest
'        bidjob.LinkRequest

 '   Case Is = 3
        'transfer these bidding objects to the outgoing factory controller text box's
  '      bidcycletime2.Text = bidcycletime.Text
   '     bidloadunloadtime2.Text = bidloadunloadtime.Text
    '    bidmachinetype2.Text = bidmachinetype.Text
     '   bidpartno2.Text = bidpartno.Text
      '  bidsetuptime2.Text = bidsetuptime.Text
       ' bidtoolstype2.Text = bidtoolstype.Text
        'bidiffile2.Text = bidiffile.Text
        'bidjob2.Text = bidjob.Text

'    Case Is = 4
        'poke all the outgoing bidding objects to the factory controller
 '       bidcycletime2.LinkPoke
  '      bidloadunloadtime2.LinkPoke
   '     bidmachinetype2.LinkPoke
    '    bidpartno2.LinkPoke
     '   bidsetuptime2.LinkPoke
      '  bidtoolstype2.LinkPoke
       ' bidiffile2.LinkPoke
        ' bidjob2.LinkPoke

'    Case Else
        'do nothing


'End Select




End Sub

Sub bidtime_Change ()

timer2.Interval = bidtime.Text
End Sub

Sub Form_Load ()
```

BID_ROTA.FRM - 3

```
'This application was written by
'Michael Butler
'Department of Production Technology
'New Zealand
'Copyright 1994

'This application is designed to rotate a bidding token around the nodes that consist of the
factory
'this is done by first testing to see if any other copies of bidrota are operating by testing
 to see
'if the bid operate block is 1, if it is zero then it sets the bid operate to 1 then starts m
oving the bidtoken
'around the factory.  If a node has something to auction then it will take the bidding token
by setting the bid enable
'block to 1 for the duration of the bidding process.
'
'The second function of this application is to transport Bidding requests and the bidding obj
ects between the current
'cell and the factory controllers.
'The Bidrequest block is the control block in this case
'If 1 is entered then the bidrequest is transported from the source node to the factory contr
oller
'If 2 is entered then the bidding objects are loaded from the source node into this applicati
on
'If 3 is entered then the data is transfered from the current source links to the factory lin
ks
'If 4 is entered then the bidding objects in the factory links are transported to the factory
 controller


On Error Resume Next

' Create new file specs
filenum = FreeFile
position = 1

'sets timer interval into screen
bidtime.Text = timer2.Interval

'open DDE links between this bidrotation program and the factory controller

'Call CommDDELink(Factory_Controller & ".bidrequest.A_CV", bidrequest2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidcyctm.A_CV", bidcycletime2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidloadun.A_CV", bidloadunloadtime2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidmachtyp.A_CV", bidmachinetype2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidpartno.A_CV", bidpartno2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidsetupm.A_CV", bidsetuptime2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidtooltyp.A_CV", bidtoolstype2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidiffile.A_CV", bidiffile2, Link_Manual)
'Call CommDDELink(Factory_Controller & ".bidjob.A_CV", bidjob2, Link_Manual)

' Opening file machine list file
Open defaultpath & "nodefile.txt" For Random Shared As filenum Len = Len(NodeListfile)

position = 1

'Do until reachs end of file
If Not EOF(filenum) Then

    ' This is to place contents of file into list on screen
    Get filenum, position, NodeListfile

    Do Until EOF(filenum)

        'display to error text box
        errortext.Text = Error

        If Trim(NodeListfile.node) Like "???#" Then

           'if record is not empty then setup links for bidding enable
            Call CommDDELink(NodeListfile.node & ".bidenable.A_CV", bidenabletext, Link_Manua
l)

            'Test if another application is already operating
            bidenabletext.LinkRequest
            If bidenabletext.Text = 0 Then

                'if enable is 0 or if no link is made then this case is true
```

```
                ' set the bidenable to high to let the nodes know that a bodrota application
is operating
                bidenabletext.Text = 1
                bidenabletext.LinkPoke

                'Close DDE link to this DBB
                bidenabletext.LinkMode = none

                Else
                    'another bidrota application must be operating goto end and close all lin
ks
                    End
                    End If

            End If

            position = position + 1
            Get filenum, position, NodeListfile

            'safety trip
            If position = 20 Then
                MsgBox "File acces error in nodefile.txt"
                End
                End If


        Loop
End If

'set position to 1 again
position = 1

'close file again
Close filenum

'display to error text box
errortext.Text = Error

End Sub

Sub mnuabout_Click ()
MsgBox "This application was Designed by Michael Butler, Copyright 1994.  It was designed to
pass a Token around the Virtual Factory Database Blocks allowing each cell controller to have
 a chance to become the center of Bid decision making.  The token is passed around in a block
 called BidToken.F_CV.  The Bidenable.F_CV block goes to 1 if this application is operating.
 The block Bidoperate.F_CV is used by the cell to establish if it wishes to use the bid tiken
 and make a bid.", MB_ICONEXCLAMATION

End Sub

Sub mnuhowto_Click ()
MsgBox "This module is the control for the bidding object transportation.  If 1 is entered th
en the bidrequest is transported from the source node to the factory controller.  If 2 is ent
ered then the bidding objects are loaded from the source node into this application.  If 3 is
 entered then the data is transfered from the current source links to the factory links.  If
4 is entered then the bidding objects in the factory links are transported to the factory con
troller", MB_ICONEXCLAMATION

End Sub

Sub Timer2_Timer ()
'
'This application was written by
'Michael Butler
'Department of Production Technology
'New Zealand
'Copyright 1994

' This program is the bidding rotation application
' when this program starts it sets up links with all the nodes specified in the nodefile.txt
file
' the first link is to bidenable.a_cv this lets all the nodes know that a copy of this applic
ation is operational
' This appliation then works through the nodefile list giving each node one to place chance t
o bid
'

On Error GoTo errorhandle
```

BID_ROTA.FRM - 5

```
' Create new file specs
filenum = FreeFile

' Opening file machine list file
Open defaultpath & "nodefile.txt" For Random Shared As filenum Len = Len(NodeListfile)

'display to error text box
errortext.Text = Error

'disable timer so it is unable to respone to timer event
timer2.Enabled = False

On Error Resume Next

'request data
bidoperatetext.LinkRequest
bidtokentext.LinkRequest

'This checks to see if bidtoken is passed and if bid token has been taken if so then wait for
 bid token to be returned, or for 60 revolutions to be performed
    If CInt(bidoperatetext.Text) <> 1 Or CInt(bidtokentext.Text) <> 1 Or counter = 60 Then
        'assume that node has nothing to bid, or finished bid and rotate bidding token to nex
t node

        'reset bidding token
        bidtokentext.Text = 0
        bidtokentext.LinkPoke

        'close dde link so able to establish new links to next destination
        bidtokentext.LinkMode = none
        bidoperatetext.LinkMode = none

        'close all bidding object DDE links
        bidrequest.LinkMode = none
        bidcycletime.LinkMode = none
        bidloadunloadtime.LinkMode = none
        bidmachinetype.LinkMode = none
        bidpartno.LinkMode = none
        bidsetuptime.LinkMode = none
        bidtoolstype.LinkMode = none


        Get filenum, position, NodeListfile

        If Not EOF(filenum) Then

            Get filenum, position, NodeListfile
            'set DDE links for bidding token and bidding operational
            Call CommDDELink(NodeListfile.node & ".bidtoken.A_CV", bidtokentext, Link_Manual)
            Call CommDDELink(NodeListfile.node & ".bidoperate.A_CV", bidoperatetext, link_aut
omatic)

            'open all bidding object link for current node and this application
            Call CommDDELink(NodeListfile.node & ".bidrequest.A_CV", bidrequest, link_automat
ic)
            Call CommDDELink(NodeListfile.node & ".bidcyctm.A_CV", bidcycletime, Link_Manual)
            Call CommDDELink(NodeListfile.node & ".bidloadun.A_CV", bidloadunloadtime, Link_M
anual)
            Call CommDDELink(NodeListfile.node & ".bidmachtyp.A_CV", bidmachinetype, Link_Man
ual)
            Call CommDDELink(NodeListfile.node & ".bidpartno.A_CV", bidpartno, Link_Manual)
            Call CommDDELink(NodeListfile.node & ".bidsetupm.A_CV", bidsetuptime, Link_Manual
)
            Call CommDDELink(NodeListfile.node & ".bidtooltyp.A_CV", bidtoolstype, Link_Manua
l)
            Call CommDDELink(NodeListfile.node & ".bidiffile.A_CV", bidiffile, Link_Manual)
            Call CommDDELink(NodeListfile.node & ".bidjob.A_CV", bidjob, Link_Manual)


            position = position + 1
            'set bidding token for current node to enable
            bidtokentext.Text = 1
            bidtokentext.LinkPoke
            counter = counter + 1

            bidnode.Text = NodeListfile.node

                Else
                    'At end of file reset counter
```

```
                    position = 1
                     End If

    Else
        'node has taken bid token and is now bidding must wait until finished this bid before
 moving on.

        timer2.Enabled = True
        Close filenum
        Exit Sub

        End If

'exit this sub and wait for next round
timer2.Enabled = True

'display to error text box
errortext.Text = Error


'close file again
Close filenum

'this part tests if 15 rotations have passed if so then the bid rotation program will be clos
ed down
' with all bidoperate blocks being set to 0, this will allow other bidoperate blocks to start
up
'If counter = 15 Then
  '  Call resetoperate
   ' End
    'End If


errorhandle:

    If Err = 52 - 76 Then

        Err = 0
        Close filenum
        'try gain
        End If
        Resume
End Sub
```

MAINPROG.BAS - 1

```
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'Define a global variable to store status of user.
'This value will be Peasant, Supervisor, or Manager.

Global User

'Define a variable to determine whether or not FIX is running
'Values will be "Yes" or "No"

Global fixrunning, DDERunning, DBaseRunning

Global cell 'current cell no

'***This sets the factory controller in the factory***
Const factory_Controller = "cim2"


'Variables for every cell position:
Global cell1TL, cell1TR, cell1BL, cell1BR
Global cell2TL, cell2TR, cell2BL, cell2BR
Global cell3TL, cell3TR, cell3BL, cell3BR
Global cell4TL, cell4TR, cell4BL, cell4BR
Global cell5TL, cell5TR, cell5BL, cell5BR
Global cell6TL, cell6TR, cell6BL, cell6BR
Global cell7TL, cell7TR, cell7BL, cell7BR
Global cell8TL, cell8TR, cell8BL, cell8BR
Global cell9TL, cell9TR, cell9BL, cell9BR

' ***Definitions for job and task selection***
'
' This type contains all the fields/ information required about a particular job

Type jobrecord

    jobname As String * 13
    currentindex As Integer
    Finaltask As Integer
    tasknumber As Integer
    Machinetype As Integer
    taskname As String * 13
    Estcycletime As Integer
    setuptime As Integer
    Estloadunloadtime As Integer
    Noofcycles As Integer
    Machinefilename As String * 13
    Geometricfilename As String * 13
    Toolsrequired As Integer
    Fixturesfilename As String * 13
    Status As Integer
    duedate As String * 8
    force As Integer

End Type

' Define job record type and global file management variables
Global Recordname As jobrecord  'Record structure
Global temprecord As jobrecord

' Status variables for file manipulation
Global filestatus As Integer
Global isfilenew As Integer
Global filepath As String
Global position As Integer
Global filenum As Integer

' Constants for file usage
Global Const filehasbeenopened = 1
Global Const filehasbeenclosed = 0
Global Const defaultpath = "r:\fix\"
Global Const newfile = 1
Global Const modifyfile = 2
'***End***
'
'
```

```
'***Definitions for machine selection file type and ascociated parts (*.mch) ***
'
' Type definitions
Type machinerecord
    Machinename As String * 14
    currentindex As Integer
End Type
'
' file manipulation variables
Global machinefilenum As Integer
'***End***
'
'
'***Definitions for Date record file type and related variables
'
'Type definition
Type daterecord
    Filename As String * 13
    Date As String * 8
End Type
'
'File manipulation variables
Global datefilenum As Integer
Global datefile As daterecord
'***End***
'
'
'***Definitions for node record type, this type contains all the information about the cells
in a particular node.
'
Type noderecord

    nodename As String * 10
    Devicename As String * 20
    Devicetype As Integer
    setuptime As Integer
    downtime As Integer
    serviceinterval As Integer
    IFFiletype As Integer
    CommunicationID As Integer
    communicationparameters As String * 40
    Partservicetype As String * 15
    feedbacksignal As String * 100
    communicationsport As Integer

End Type
'
'***END***
'

'***Definitions for node list record type
'
Type nodelistrecord
    node As String * 10

End Type
'***END***



Sub readfromfile (record As jobrecord)
'read from file at position into recordname
If position = 0 Then
    position = 1
    End If

Get filenum, position, record

End Sub

Sub savetofile (record As jobrecord)

Dim response As Integer

' save all information to file
response = MsgBox("This task has been modified do you wish to save it?", MB_YESNO, "Save Job"
)
    If response = IDYes Then
```

```
    ' save file
     Put #filenum, position, record
  End If
    ' No save file
 End Sub

Sub testfix ()

'This function is designed to warn users that Fixdmacs must be operating to perform some of t
he functions in this application

Dim response, response2 As Integer                    'Variable to see if Fix is running
                                'Check if Fix is running

If fixrunning = IDYes Then

    'This is fine
    Exit Sub

    Else
    'fixrunning = IDNo or something else

    response = MsgBox("Is FIX DMACS currently running.", 4, Title)

    If response = IDNo Then
        response = MsgBox("This operation requires Fixdmacs to be operating.  Do you wish to
continue.", 4, Title)

        If response2 = IDNo Then
            'end and pass out that this operation will not continue

            fixrunning = IDNo
            Exit Sub
            End If

         MsgBox "Before continuing on with this operation you must first Switch to program ma
nager and start Fixdmacs"

         ElseIf response = IDYes Then
            'then fix is running

            fixrunning = IDYes
         End If

    End If

End Sub
```

QUEUE1.BAS - 1

```
Option Explicit
'This application was written By:
'Michael Butler
'Department of Production Technology
'Massey University
'Copyright 1994

'**For queue program
'** I dont know why this works but it does?***
Type queue

    Job_Name As Variant
    IF_Filename As Variant
    No_ofparts As Variant
    Setup_Time As Variant
    Tools_Required As Variant
    Machine_Type As Variant
    Cycle_Time As Variant
    Loadunload_Time As Variant

End Type

Global Const factory_Controller = "cim2"


' MsgBox parameters
Global Const MB_OK = 0                    ' OK button only
Global Const MB_OKCANCEL = 1              ' OK and Cancel buttons
Global Const MB_ABORTRETRYIGNORE = 2      ' Abort, Retry, and Ignore buttons
Global Const MB_YESNOCANCEL = 3           ' Yes, No, and Cancel buttons
Global Const MB_YESNO = 4                 ' Yes and No buttons
Global Const MB_RETRYCANCEL = 5           ' Retry and Cancel buttons

Global Const MB_ICONSTOP = 16             ' Critical message
Global Const MB_ICONQUESTION = 32         ' Warning query
Global Const MB_ICONEXCLAMATION = 48      ' Warning message
Global Const MB_ICONINFORMATION = 64      ' Information message

Global Const MB_APPLMODAL = 0             ' Application Modal Message Box
Global Const MB_DEFBUTTON1 = 0            ' First button is default
Global Const MB_DEFBUTTON2 = 256          ' Second button is default
Global Const MB_DEFBUTTON3 = 512          ' Third button is default
Global Const MB_SYSTEMMODAL = 4096         'System Modal

' MsgBox return values
Global Const IDOK = 1                     ' OK button pressed
Global Const IDCANCEL = 2                 ' Cancel button pressed
Global Const IDABORT = 3                  ' Abort button pressed
Global Const IDRETRY = 4                  ' Retry button pressed
Global Const IDIGNORE = 5                 ' Ignore button pressed
Global Const IDYES = 6                    ' Yes button pressed
Global Const IDNO = 7                     ' No button pressed


Function loadApplication (Appname, Path) As Variant

'Written by Michael Butler
'Department of Production Technology
'Massey University
'
'This function takes an application name and path then attempts to start it
'if any errors occur it aborts, returns loadapplication yes or no ie it has been loaded
'
'
'Improvement can be made here by testing if the application is open before it is opened using
 an API call
'

Dim response, x                                    'response from message box
                            'return from message box

    On Error GoTo ErrorHandel   'if an error occurs goto errorhandel
    x = Shell(Path & Appname)
    loadApplication = True

Exit Function

ErrorHandel:
```

```
    response = MsgBox("Is " & Appname & " currently running?", MB_YESNOCANCEL, "Application S
tatus")       'Ask if want to start application
    If response = IDYES Then
        loadApplication = True
    Else
        MsgBox "Error in opening application, this operation may not continue", 48, "Applicat
ion Status"
        loadApplication = False
    End If
    Exit Function

End Function
```

# Appendix L

## Data and Experimental Method for Experiment 5

This appendix contains the data for experiment number 5; The effect of scan time upon the performance of the bidding algorithm.

## The Effect of Scan Time upon the Bidding Algorithm

The effect of database scan time on the operation of FIX DMACS program block was investigated through experimental methods. By altering the scan time then determining how long the Bidding algorithm took to operate it was possible to determine two things.

1. The effect of the scan time had upon the speed of operation of program blocks within FIX DMACS.

2. The range of operational speeds the bidding algorithm was able to operate over effectively. Thereby determining the maximum speed of operation for the bidding algorithm.

The program cycle of the bidding algorithm consists of five major identifiable points. Each of these points of operation where used in accessing the speed of operation of the entire bidding algorithm. The time taken for the bidding algorithm, to move from one point to the next was accessed through manual timing.

Table showing the five different identifiable points during the bidding algorithm program cycle.

| | |
|---|---|
| 1 | Bid Request is received |
| 2 | Bid is calculated |
| 3 | Bid is placed |
| 4 | Bid Acknowledgment is sent |
| 5 | Bid blocks are reset |

The different times used to gauge the speed of operation were:

$T_1 = 1$ to 2

$T_2 = 2$ to 3

$T_3 = 3$ to 5

A number of experiments were conducted in order to access the time taken for each of the about times these were then added to give the total time taken for the bidding algorithm to operate. These experiments where undertaken by altering the scan time to a certain value then running the bidding algorithm and determining $T_1$ to $T_3$. This was repeated ten times for each scan time value.

| Scan Time (sec) | $T_1$ (sec) | $T_2$ (sec) | $T_3$ (sec) | $T_{Total}$ (sec) |
|---|---|---|---|---|
| 100 | 17 | 38 | 73 | 128 |
| 10 | 9 | 20 | 29 | 58 |
| 8 | 6 | 14 | 24 | 44 |
| 6 | 6 | 12 | 18 | 36 |
| 4 | 3 | 11 | 13 | 27 |
| 2 | 2 | 4 | 8 | 14 |
| 1 | 1 | 2 | 6 | 9 |
| 0.5 | 1 | 1 | 6 | 8 |

From this experiment the minimum value was determined as 8 seconds for the operation of the bidding algorithm. It is possible to take the scan time of the database down to 0.1 seconds. However it the minimum scan time which could be accurately accessed was 0.5, by extrapolation it would be fair to say that the minimum possible speed of operation of the bidding algorithm is approximately 7 seconds, at a scan time of 0.1 seconds.

# Appendix M

## Data and Experimental Method for Experiment 1,2 & 4

This application contains data and experimental method for experiment numbers;

1 The effect of bias within the Bidding Algorithm.
2 Evaluating the functionality of the Bidding algorithm.
4 The evaluation of the performance of the EFT Bidding algorithm.

## Simulation of the Auction Algorithm

A number of simulations were undertaken to evaluate various aspects of the auction algorithm. These simulations consisted of three experiments. The first experiment was designed to demonstrate the functional bias of the auction algorithm. The second experiment was designed to evaluate the functionality of the auction algorithm. The third and final experiment was done in order to evaluate the effectiveness of the auction algorithm in comparison to other bidding algorithms.

These simulations were undertaken through examining the operation of the bidding algorithm at high speed, ie with the scan time set to 1 second. The algorithm itself was studied to access its functionality. In the case of the first two experiments, this same functionality was then applied to manual simulations. The third experiment was conducted using computer software which simulated the functionality of the auction algorithm as it would operate in a real situation within FIX DMACS.

The consistent operation of the auction algorithm meant that, if an experiment was repeated twice with the exact same conditions, the system response would be identical. It was also determined that the response of the system to unexpected change, could be anticipated. When cells or devices are added or subtracted from the system no change in the timing or operation of the bidding algorithm occurs. Thus two assumptions were made in respect to manual simulation of the factory.

1. The functionality of the bidding algorithm would not change over time.

2. The actual speed of operation of the bidding algorithm compared to the time taken for a cell to process a part would be so disproportionate that the time taken to bid for the part could be negated altogether.

These two assumptions meant that a realistic simulation of the operation of the bidding algorithm could be conducted, so as to determine its effectiveness when operating with a scan time of 1 second.

## Experiment 1

This experiment was conducted to demonstrate the bias in the decision making process of the auction algorithm. This bias is consistent and means that the certain cells are favoured over other cells when the result of the auction is tied. From experimentation upon the operating system this case is a relatively common occurrence, especially in a under utilised system where often two cells will place a bid of zero. This means that if two cells place the same bid for a task, one particular cell will always win the task, and thus will have a higher machine utilisation factor than the other two cells. This bias was demonstrated through conducting a simple manual simulation using the same bidding algorithm as was implemented within the FMS control system. A number of tasks $T_1$ to $T_{10}$ were created with a random length and then were release in to the theoretical factory at time T=0. These jobs were distributed amongst the devices which were all of the same type, through the use of the EFT bidding algorithm, as shown below. The EFT bidding algorithm used in this experiment was used with transportation time and auction length set to 0.

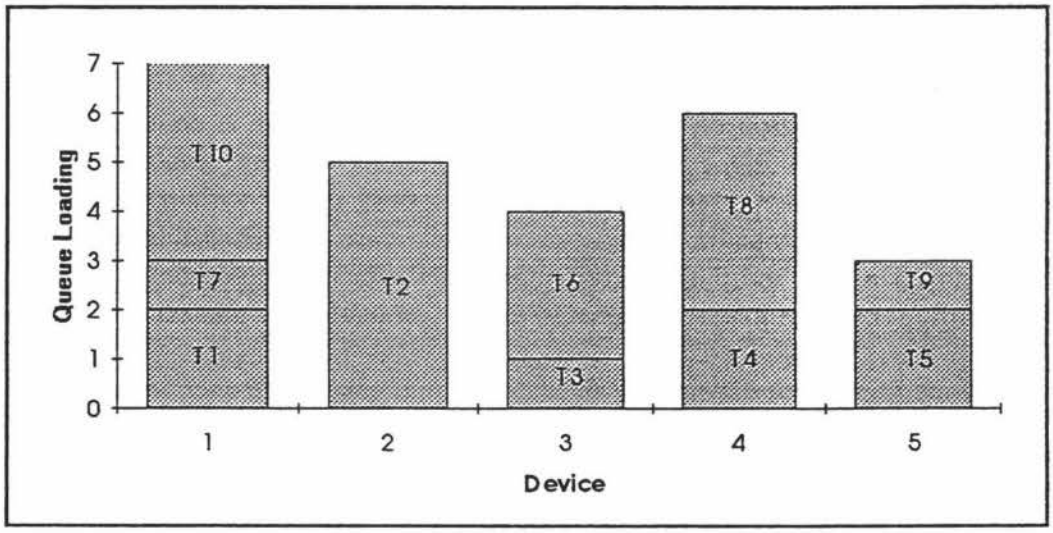



Figure showing queue loading over five devices

This experiment shows five devices all of the same type. Ten jobs were released into this factory and where distributed amongst five different devices. This diagram shows the bias towards the devices on the left hand side. This is because when the bidding algorithm is making the decision to allocate the job if two cells/devices have placed an identical bid then the cell/device with a lower number will always win the bid. Thus a bias occurs towards the cells/devices with lower identification numbers, ie devices on left hand side of figure.

## Experiment 2

In this experiment the functionality of the bidding algorithm was demonstrated. This was achieved by devising a theoretical factory and then releasing a number of jobs into the factory over a period of time. The operation of the auction algorithm in this simulation was identical to the physical system. Thus the aim of this experiment was to access how the distribution of jobs would be achieved by the bidding algorithm.

Two simulations were conducted in this experiment. The first simulation was undertaken to access the distribution of jobs amongst the different cells within the factory. This factory consisted of four cells, with each cell having either three or four devices, each device within a cell was different. Their were six different types of devices in the factory. Eleven different jobs were released into the factory over a period of time, each job consisted of between one and five tasks. Each task was required to be operated upon by one type of device for a minimum of one time unit and a maximum of 10 time units. When the first task in a job was completed then an auction was held to determine which cell would undertake the next task, if there was one.

Simulation 1

The first simulation operated as an under utilised system, ie more than 50% of the duration of the simulation their was over production capacity. This simulation demonstrated the operation of the EFT auction algorithm.

The simulation conducted was of a factory consisting of four cells, each cell had up to four machines, there where five different types of machines available in the factory. The choice of which of the five different types of machines would be placed in each cell was done randomly. In the case of the first simulation, eleven randomly created jobs were released into the system for processing, using the EFT bidding algorithm. All the jobs were released at time $T = 0$.

The data for this simulation is shown at the end of this appendix, the results of this simulation are shown in figure 6.2, in chapter 6.

Simulation 2

The second simulation conducted was under the same conditions as the first, however the aim of this simulation was to demonstrate the functionality of the auction algorithm in a higher load situation. This experiment was not conducted as an investigation into the performance of the EFT bidding algorithm. A better indication of the functionality of the bidding algorithm was able to be gained in the case where their was more work to be done than machines to do the work, ie in the case where buffering or queuing of jobs occurred. This was achieved through using the same set up as used in the previous simulation, however rather than eleven, eighteen jobs were used and were released in three lots, separated by five time units. This provided enough jobs to allow queuing at all machines to occur.

The data for this simulation is shown at the end of this appendix, the results of this simulation are shown in figure 6.3, in chapter 6, the jobs used in this simulation were the same as those used in the first simulation.

Some evaluation of the results of this simulation was conducted, however it was felt that a number of repetitions would need to be conducted before any statistically sound information could be gained from this simulation.

The evaluation method used was the same as that employed by Michael Shaw (Shaw 1987). This was used by Shaw to compare the effectiveness of a number of different scheduling algorithms. This allowed the simulation to be compared in a general sense to the simulations conducted by Shaw. However it was realised that a direct comparison could not be made as the experimental procedure used by Shaw could not be recreated.

The evaluation of the experiment gave the following results. The evaluation methods are provided in the following section.

| | |
|---|---|
| Percentage of jobs late | 30% |
| Average lateness of jobs | 3.3 |
| Mean flow time | 16.6 |

## Experiment 3

In this experiment the effectiveness of the bidding algorithm was evaluated. This was achieved through the simulation in computer software of a factory, similar to that which would be achievable within the real FIX DMACS system. The simulation was designed so that the functionality would be as close to a real implementation of the EFT auction algorithm as possible. Thus this simulation provided a worthwhile insight into the effectiveness of the EFT bidding algorithm. This simulation was designed and undertaken by other researchers within the department of Production Technology (Ginting 1994b).

This simulation was undertaken through the construction of a computer based simulation model. This model was created to as closely represent the operation of the auction algorithm within the FMS control system as possible. The model of the auction algorithm used in this simulation, considered a more realistic assessment of EFT. Whilst in the FMS auction algorithm transportation time is assumed to be zero, in this simulation the transportation time was set to a constant. The computer simulation itself was written within Visual Basic. The scheduling of jobs was simulated within a factory consisting of four cells, one cell containing two machines, another containing three machines and the other two containing four machines. Data from this simulation was collected after it had been verified that the model had achieved a steady state. The simulation was then run 12 number of times whilst in the steady state condition, in order to assess the effectiveness of the EFT auction algorithm.

The evaluation method used was the same as that employed by Michael Shaw (Shaw 1987). This was used by Shaw to compare the effectiveness of a number of different scheduling algorithms. It was realised that a direct comparison could not be made as the experimental procedure used by Shaw could not be recreated.

The evaluation method used by Shaw consisted of four different measures, three of these were used in the evaluation of the simulation; Percentage of jobs late, average lateness of jobs and mean flow time.

These measures were calculated from the comparison of a jobs due date to its actual arrival date. The due date was calculated from the simulation using the a similar method to Shaw. The formula that Shaw (Shaw 1987) uses is the same as below except Shaw does not take into account time taken in transportation.

$$Due\_Date = TNOW + (Est\_TPT) \times 1.3 + (NOp) \times SD + TD$$

Due_Date — This is the time which the job should be completed by.

TNOW — Time Now, in the case of this simulation, this was the time at which the job was released into the system.

Est_TPT — Estimated total processing time, this is the total time taken for the job to be processed.

NOp — Number of operations, this is the number of different operations performed by the factory upon the job.

SD — Standard Delay, this is a time delay which combines a communications delay with the auction evaluation delay time. This was done to make the evaluation more realist.

TD — Transportation Delay, this is the time which the is required to transport the work piece to the place of processing. This was estimated with a constant.

This formula took the total processing time relative to the time the job was released then multiplied this by 1.3 plus the number of operations performed multiplied by a standard delay, then added the transportation delay.

The simulation shown at the end of this appendix was undertaken and the following measures were taken.

Percentage of jobs late $$\frac{\sum \text{Late jobs}}{\text{Total number of jobs}} \times 100$$

Average Lateness $$\frac{\sum \text{Lateness of late jobs}}{\sum \text{Late jobs}}$$

Mean flow time $$\frac{\sum \text{Job process time}}{\text{Total number of jobs}}$$

| | Shaw 1987 | | | Simulation 3 |
|---|---|---|---|---|
| | Bidding EFT | Bidding SPT | Myopic SPT | Sim Bidding EFT |
| Percentage of jobs late | 20.28 [1] | 33.90 [1] | 36.76 [1] | 56.94 |
| Average lateness (AL) | 3.01 [1] | 3.24 [1] | 4.08 [1] | 4.41 |
| Mean flow time (MFT) | 20.79 [1] | 21.97 [1] | 22.14 [1] | 34.13 |

([1] These figures are representative of experimental data from Shaw 1987)

It was realised however that simulation 3 and the results collected from Shaw could not be effectively compared, as Shaw's experimental method and some of his basic assumptions are not known. However in order to make simulation 3 as realistic as

possible a few adjustments were made to the known method of Shaw. These factors were felt to contribute to the significantly different results obtained by Ginting and Shaw.

## Experiment 2

## Simulation 1

### SIMULATION DATA

Simulation Rules

1. Job release order 1-11

2. If two cells place the same bid for the a job then the cell with the lowest cell number will obtain the job

3. Time moves from left to right, time starts at T = 0

4. A **Bold** number indicates the successful bidding for a job

| | | Bids | | | |
|---|---|---|---|---|---|
| **Bid Number** | **Job Number** | **Cell 1** | **Cell 2** | **Cell 3** | **Cell 4** |
| 1 | 1 | **0** | 0 | 0 | 0 |
| 2 | 2 | **0** | | | |
| 3 | 3 | | **0** | | 0 |
| 4 | 4 | **0** | 0 | 0 | |
| 5 | 5 | 7 | **0** | 0 | |
| 6 | 6 | | 14 | | **0** |
| 7 | 7 | 7 | 3 | **0** | |
| 8 | 8 | 7 | **0** | 0 | |
| 9 | 9 | 7 | 5 | **0** | 0 |
| 10 | 10 | **0** | 0 | 0 | 0 |
| 11 | 11 | **17** | | | |
| 12 | 5 | **19** | | | |
| 13 | 7 | 3 | 2 | **0** | 0 |
| 14 | 9 | | 11 | | **7** |
| 15 | 8 | **0** | 0 | 0 | 0 |
| 16 | 4 | **0** | 0 | 0 | 0 |
| 17 | 1 | **0** | 0 | 0 | |
| 18 | 1 | 14 | **0** | 0 | 0 |
| 19 | 6 | 14 | 4 | **0** | 0 |
| 20 | 6 | **0** | 0 | 0 | 0 |
| 21 | 3 | **0** | 0 | 0 | 0 |
| 22 | 1 | | **0** | | 0 |
| 23 | 9 | 7 | **0** | 0 | 0 |
| 24 | 11 | **0** | 0 | 0 | |
| 25 | 5 | **0** | 0 | 0 | **0** |
| 26 | 11 | 4 | **0** | 0 | 0 |
| 27 | 5 | 3 | 0 | | 0 |

## Experiment 2

## Simulation 2

### SIMULATION DATA

Simulation Rules

1. Job release order 1-18

2. If two cells place the same bid for the a job then the cell with the lowest cell number will obtain the job

3. Time moves from left to right, time starts at T = 0

4. A **Bold** number indicates the successful bidding for a job

| | | Bids | | | |
|---|---|---|---|---|---|
| **Bid Number** | **Job Number** | **Cell 1** | **Cell 2** | **Cell 3** | **Machine Type** |
| 1 | 1 | | **0** | 0 | A |
| 2 | 2 | **0** | 0 | | C |
| 3 | 3 | | **0** | 0 | E |
| 4 | 4 | **0** | | **0** | B |
| 5 | 5 | 7 | | **0** | B |
| 6 | 6 | 14 | | **0** | E |
| 7 | 7 | 7 | | **3** | B |
| 8 | 8 | | 7 | **0** | A |
| 9 | 9 | | 7 | **5** | A |
| 10 | 10 | **0** | 0 | 0 | D |
| 11 | 11 | 17 | **0** | | C |
| 12 | 12 | | **7** | 8 | A |
| 13 | 13 | 7 | | **6** | B |
| 14 | 5 | 14 | **2** | | C |
| 15 | 14 | **0** | 0 | 0 | D |
| 16 | 15 | 3 | **0** | 0 | D |
| 17 | 16 | 3 | 3 | **0** | D |
| 18 | 8 | **3** | 3 | 3 | D |
| 19 | 7 | | 8 | **2** | A |
| 20 | 11 | **2** | | 4 | B |
| 21 | 4 | 3 | **1** | 1 | D |
| 22 | 1 | 4 | | **2** | B |
| 23 | 5 | | 6 | **4** | A |
| 24 | 9 | | **6** | 6 | E |
| 25 | 13 | 8 | **0** | | C |
| 26 | 17 | 7 | **2** | | C |
| 27 | 18 | **7** | 7 | | C |

| | | | | | |
|---|---|---|---|---|---|
| 28 | 11 | **0** | 14 | 0 | D |
| 29 | 13 | | **2** | 3 | A |
| 30 | 1 | 4 | 13 | **0** | D |
| 31 | 3 | | 3 | **1** | A |
| 32 | 12 | **0** | | 0 | B |
| 33 | 6 | **2** | 11 | 2 | D |
| 34 | 5 | 2 | 10 | **1** | D |
| 35 | 1 | | 2 | **0** | E |
| 36 | 6 | **0** | | 0 | B |
| 37 | 12 | **0** | 8 | 4 | D |
| 38 | 17 | 1 | | **0** | B |
| 39 | 13 | **4** | 8 | 4 | D |
| 40 | 12 | | **0** | 0 | E |
| 41 | 5 | | 5 | **0** | E |
| 42 | 17 | 5 | 4 | **0** | D |

## CALCULATION OF MEASUREMENT INDICIES

| Release Time | Job Number | Job length | Minimum Completion Time | Expected Completion Time | Actual Completion Time | Lateness of Job |
|---|---|---|---|---|---|---|
| | 1 | 19 | 19 | 25 | 21 | |
| | 2 | 19 | 19 | 25 | | |
| | 3 | 28 | 28 | 27 | 19 | |
| | 4 | 25 | 25 | 32 | 25 | |
| 1st | 5 | 19 | 19 | 25 | 26 | 1 |
| Release | 6 | 12 | 12 | 16 | 18 | 2 |
| T = 0 | 7 | 7 | 7 | 10 | 12 | 2 |
| | 8 | 7 | 7 | 10 | 10 | |
| | 9 | 7 | 7 | 10 | 18 | 8 |
| | 10 | 3 | 3 | 4 | 3 | |
| | 11 | 14 | 14 | 19 | 16 | |
| | 12 | 19 | 19 | 25 | 26 | 1 |
| | 13 | 19 | 19 | 25 | 31 | 6 |
| 2nd | 14 | 3 | 8 | 9 | 8 | |
| Release | 15 | 3 | 8 | 9 | 8 | |
| T = 5 | 16 | 3 | 8 | 9 | 8 | |
| 3rd | 17 | 14 | 24 | 29 | 26 | |
| T = 10 | 18 | 19 | 28 | 35 | 31 | |

## Experiment 3

## Simulation 1

The results of the experiment were calculated from data (Ginting 1994b) collected from the simulation of a factory using a EFT scheduling algorithm. The results of this experiment were collected whilst the simulation was operating within a steady state. The simulation parameters were as follows;

**SIMULATION PARAMETERS**

| | |
|---|---|
| MTBA | 1.49 |
| Number of repetitions | 12 |
| Number of factory controllers | 1 |
| | |
| Number of cell controllers | 4 |
| Number of machines in cell 1 | 4 |
| Number of machines in cell 2 | 3 |
| Number of machines in cell 3 | 4 |
| Number of machines in cell 4 | 2 |
| | |
| Length of each simulation | 250 hours |

**RESULTS OF SIMULATION**

| Simulation Number | Percentage of jobs late | Mean flow time (hrs) | Average lateness |
|---|---|---|---|
| 1 | 53.12 | 30.84 | 2.11 |
| 2 | 72.9 | 37.68 | 8.74 |
| 3 | 27.50 | 26.86 | -3.44 |
| 4 | 28.10 | 25.29 | -4.27 |
| 5 | 73.26 | 39.50 | 9.80 |
| 6 | 55.09 | 35.98 | 6.47 |
| 7 | 41.77 | 28.52 | -0.89 |
| 8 | 91.19 | 46.95 | 17.62 |
| 9 | 53.94 | 32.09 | 2.08 |
| 10 | 60.12 | 32.44 | 2.33 |
| 11 | 59.15 | 33.70 | 3.40 |
| 12 | 67.27 | 39.80 | 9.01 |
| | | | |
| Mean | 56.94 | 34.13 | 4.41 |

# Appendix N

## Data and Experimental Method for Experiment 3

This application contains information pertaining to experiment number 3, which was conducted in order to evaluate the operation of the auction algorithm.

## Simulation of the Auction Algorithm using the Implemented FMS Control System.

These simulations were conducted to test the correct operation of the auction algorithm operating within the experimental factory implementation of the FMS control system. These simulations were undertaken through testing the flow of a number of jobs released into the experimental factory. The factory on which the simulation was conducted, consisted of the experimental FMS control system software implemented upon three computers, two cell controllers and one factory controller. This factory communicated over a LAN.

This experiment consisted of three simulations. The job definition, release time and order was randomly selected for each job. A machine failure was introduced into simulation 1 and 2, to demonstrate how the auction algorithm recovered.

The two cell controllers each contained two simulated machines, cell 1 contained machine types 1 & 2, cell two contained machine types 1 & 3. Jobs were released into the system, the progress of the jobs being processed by the factory was then recorded. The scheduling of these jobs was undertaken using the EFT auction algorithm.

## JOB DEFINITION

Jobs were created randomly, the components of the total task time was randomly selected between 1-6. With the task cycle number randomly selected between 1-13. The tasks within each job were undertaken in order of task number. Thus task 1 had to be completed before task 2 could be started.

Table 1
Job definitions

| Job Name | Task Number | No. of cycles | Set up time | Load & unload time | Cycle no. | Machine type | Tool type | Total task time |
|---|---|---|---|---|---|---|---|---|
| Bob 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |
| | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 6 |
| | 3 | 1 | 3 | 3 | 3 | 1 | 1 | 9 |
| | | | | | | | | |
| Bob 2 | 1 | 12 | 1 | 3 | 4 | 1 | 1 | 85 |
| | 2 | 12 | 1 | 3 | 4 | 2 | 2 | 85 |
| | 3 | 12 | 1 | 3 | 4 | 1 | 1 | 85 |
| | 4 | 12 | 2 | 2 | 2 | 3 | 3 | 28 |
| | 5 | 12 | 3 | 4 | 4 | 1 | 1 | 99 |
| | | | | | | | | |
| Bob 3 | 1 | 12 | 1 | 1 | 1 | 1 | 1 | 25 |
| | 2 | 12 | 2 | 2 | 2 | 2 | 2 | 50 |
| | 3 | 12 | 3 | 3 | 3 | 3 | 3 | 75 |
| | | | | | | | | |
| Bob 4 | 1 | 3 | 19 | 20 | 1 | 1 | 1 | 82 |
| | 2 | 3 | 3 | 1 | 1 | 1 | 1 | 9 |
| | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 21 |
| | 4 | 3 | 3 | 3 | 3 | 2 | 2 | 21 |
| | | | | | | | | |
| Bob 5 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 5 |
| | 2 | 2 | 4 | 1 | 23 | 3 | 3 | 52 |
| | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 10 |
| | | | | | | | | |
| Bob6 | 1 | 4 | 3 | 1 | 2 | 2 | 2 | 19 |
| | 2 | 4 | 5 | 6 | 3 | 1 | 1 | 45 |
| | 3 | 4 | 4 | 4 | 3 | 3 | 3 | 53 |
| | 4 | 4 | 1 | 1 | 2 | 1 | 1 | 19 |

## RESULTS OF EVALUATION OF BIDDING ALGORITHM

Simulation 1

Five jobs were released from the factory controller Bob1 to Bob 5, each of the jobs consisted of a number of tasks, with each task requiring one of three machines in order for it to be processed. These jobs were all released at time T=0 or T=30.

In order to test the robustness of the bidding algorithm machine A in cell 2 was caused to fail at time T = 50. Thus this machine received no more jobs past time T = 50.

The figure below represents the results of the simulation conducted upon the FMS control system. It can be seen that the jobs flow between different machines and cells depending upon the requirements of the current task. The wide spacing between jobs in a single machine is due to the proportionally large amount of time required to hold the auction of a single job.

The release dates of the two groups of jobs were chosen arbitrarily.

Job Release dates

Job1, Job2 & Job3 T=0

Job4, Job 5 & Job3 T=30

Cell 2 machine A failed at T=50

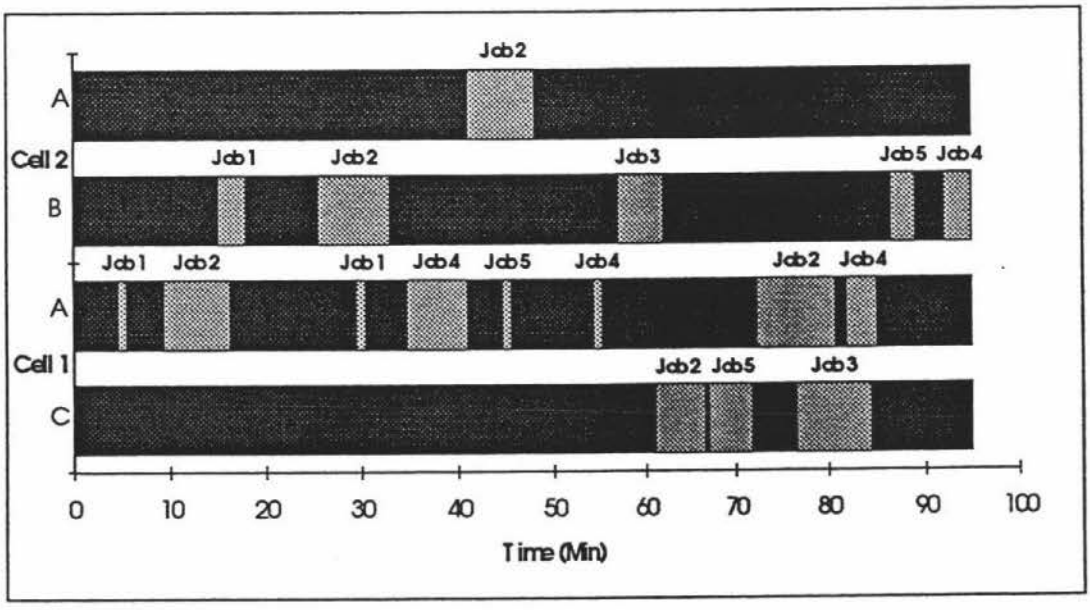



Figure 1
Results of simulation 1 using a single factory and two cell controllers of the FMS control system.

Simulation 2

Simulation 2 and three were conducted in a similar fashion to simulation one, except that both these simulations contained six jobs not five.

Job Release dates

Job5, Job4 & Job1 T=0

Job6 & Job 2 T=9

Job3 T=54

Cell 1 machine A failed at T=73

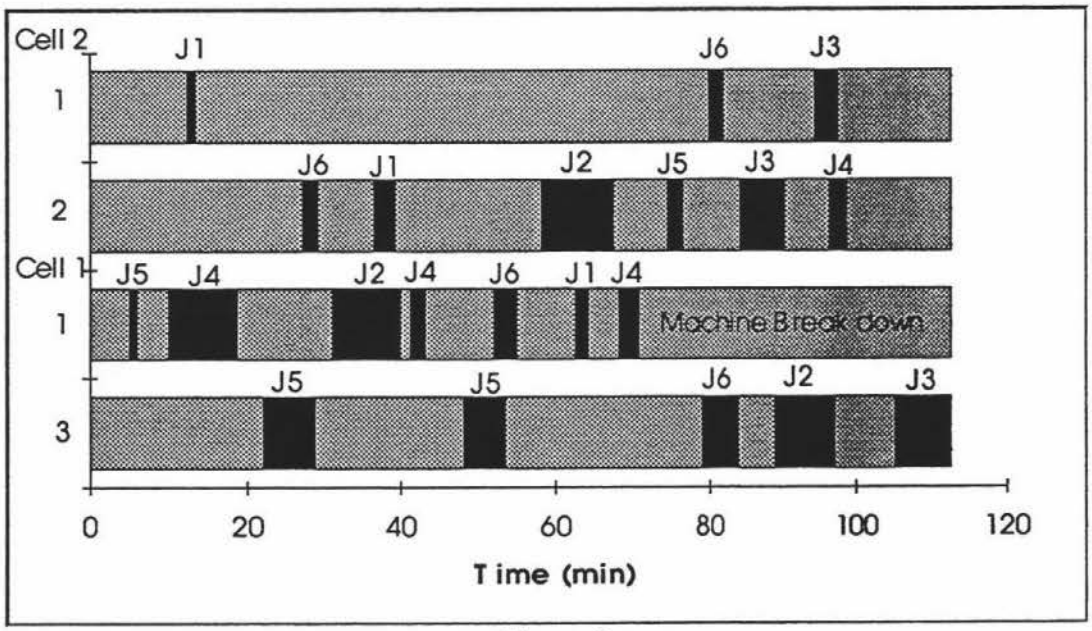



Figure 2
Results of simulation 2 using a single factory and two cell controllers of the FMS control system.

Simulation 3

Job Release dates

Job2 & Job3 T=0

Job5 & Job 1 T=28

Job6 & Job 4 T=39

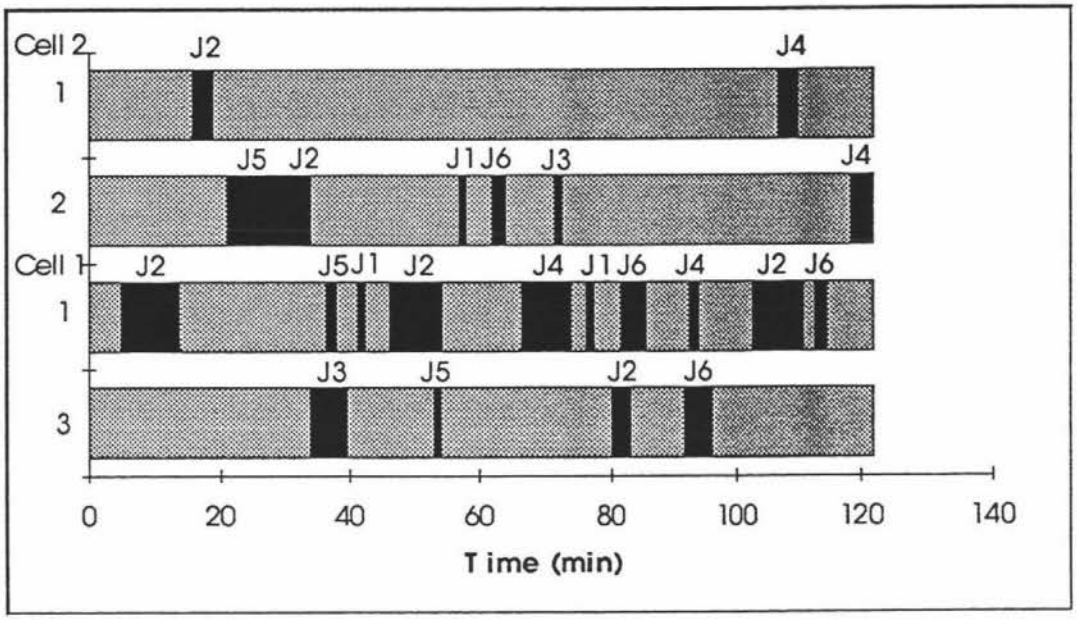



Figure 3
Results of simulation 3 using a single factory and two cell controllers of the FMS control system.

# Appendix O

## Rules Governing the Behaviour of the Intelligent Entities

## Characteristics of an Intelligent Entity

1. No master slave relationships should be developed between IE's.
2. An entity should only impart information when requested to do so.
3. An entity should not assume the co-operation of other entities.
4. Each entity must conform to the rules governing all entities as a whole.
5. Entities will be governed by rules which will define their behaviour.
6. Time critical responses should be contained within the entity.
7. Faults should be contained locally, as well as the coordination of the recovery.